Generation, Recognition And Storage

Of Structured Boolean Matrices

A Thesis Submitted
In Partial Fulfilment of the Requirements
For the Degree of

MASTER OF TECHNOLOGY

by

J.K. Navlakha

to the

Department of Electrical Engineering
Indian Institute of Technology
KANPUR

CERTIFICATE

This is to certify that the thesis entitled 'GENERATION, RECOGNITION AND STORAGE OF STRUCTURED BOOLEAN MATRICES' is a record of the work carried out under my supervision and that it has not been submitted elsewhere for a degree.

May, 1974

Dr. V. Rajaraman
Professor
Computer Centre
I.I.T. Kanpur

ACKNOWLEDGEMENTS

I express my deep sense of gratitude to Dr. V. Rajaraman for having suggested this topic and for his expert guidance, helpful suggestions and constant encouragement throughout the work. My thanks are due to Mr. A.S. Sethi for his help. I also wish to thank Mr. Kumar for the excellent job of typing the manuscript.

J.K. Navlakha

ABSTRACT

Matrices having some specified structure are called structured matrices. These are encountered in many fields like electrical engineering, structural engineering etc. This thesis deals with generation, recognition and storage of structured Boolean Matrices. The generation of these uses shift-register technique. Algorithms for recognizing different structured matrices from their row or column permuted matrices are developed. Also we have presented algorithms for recognizing three of these structured matrices from their row ~~or~~ and column permuted matrices. Another algorithm for reducing the Bandwidth of a given random matrix by possible removal of rows and columns is included.

The characterization of classes of generatable matrices based purely on their structural aspect is given. We have also given some matrix representations, e.g., Polynomial representation, Walsh-function representation etc. This was done for the purpose of identifying to which class does a matrix belong.

CONTENTS

CHAPTER 1

INTRODUCTION

As is clearly pointed out by the title, the thesis deals with generation, recognition and storage of structured Boolean matrices. The objective of the thesis is three-fold.

i) In many application areas, mentioned later, we encounter large, structured as well as sparse, matrices. Sometimes the storage requirement of these matrices is prohibitively large, resulting in our inability to solve the given problem, be it important or interesting.

In such cases, instead of storing the full matrix in the core, we take recourse to some storage technique, thereby reducing the storage requirement of the matrix. The once abandoned problem, can now be tackled easily.

ii) If the matrix in question is structured, it is highly inefficient as far as the execution time is concerned to deal with all elements of the matrix while performing arithmetic operations on it.

In such cases again, we try to reduce the computer execution time by storing the matrix utilizing its structural property, thereby avoiding as far as possible computations with 'zero' elements of the matrix.

iii) Often the structure of the matrix is not evident at the context. In such cases, recognition of one of the

structures, in the matrix is of prime importance. We will call this the 'Recognition' problem.

After a matrix is recognized to be of one of the structured forms, we wish to recover the structured form of the matrix from the seemingly random form. To develop methods for doing this was one of our prime objectives.

Literature survey revealed that though storing and indexing techniques exist for sparse matrices, nobody has paid any attention to generating, storing or recognizing methods for structured matrices. While the number of references involving sparse matrix algorithms are plenty [1, 2, 3, 8, 9, 11, 14, 18, 30, 32, 33, 41, 42, 43] ; those dealing with structured matrices are only a handful [5, 15, 25, 26, 34, 35].

We have considered six structured forms of matrices viz.,

i) Band Diagonal Matrix

ii) Band Triangular Matrix

iii) Block Diagonal Matrix

iv) 'Overlapping-blocks' Matrix

v) Multiple and 'Overlapping-blocks' Matrix, and

vi) V-shaped Matrix.

If a matrix is structured one, it is not necessary to store the full matrix in the core. In fact, we need only one row in the core at a time, and other rows can be

generated.

The chapter following the present one gives the generating techniques of 'structured matrices' using shift-registers. In this, we generate $(i+1)^{\underline{st}}$ row of the matrix from the $i^{\underline{th}}$ row, using the structural properties of the matrix. Then, at any time, storage required is proportional to N, while if the full matrix were stored; storage required is proportional to N^2. Algorithms for generating all the above mentioned structured matrices are developed and presented. Also, it is possible to recover all of them from their row or column permuted matrix. Algorithms for doing this, are also given.

It has been observed that sometimes both row and column permutations are necessary for reducing a given matrix to one of the structured forms. Chapter 3 gives three such algorithms to obtain,

i) Band Diagonal

ii) Band Triangular, and

iii) 'Overlapping-Blocks',

matrices from their row and column permuted matrix.

In some cases, it is not possible to reduce the matrix into either of the structured forms by row and column permutations. In such cases, a few rows and/or columns are to be removed and the remaining matrix converted to one of the structured forms. An algorithm for such

conversion to Block diagonal matrix exists [22,39], and a new algorithm for conversion to Band diagonal matrix is given in Chapter 4.

It is possible to generate some classes of matrices using the generating scheme of shift registers and a few allowed operations. The description of these classes, based purely on their structural aspects, appears in Chapter 5.

To characterize these classes mathematically, we tried some representations of matrices, e.g., Polynomial representation, Walsh-function representation etc. These representations are discussed in Chapter 6. It should however be mentioned that these proved to be fruitless and a method of characterizing the classes of matrices considered in this thesis, remains unsolved.

The complete work is summarized in Chapter 7, which is the concluding chapter.

To quote Tewarson [31], sparse matrices occur in the solution of many important practical problems, e.g., in structural analyses, network theory and power distribution systems, numerical solution of differential equations, graph theory, reactor diffusion as well as in genetic theory, behavioral and social sciences, and in computer programming. As the technology increases in complexity, we can expect that large sparse matrices will continue to occur in many future applications involving large systems, e.g., simulation of traffic lights, pattern recognition, urban planning and so on.

According to Pooch and Nieder, [21], it has been observed that in many of the application areas mentioned above, the matrices are not exactly random ones. In electrical network and power distribution systems, the matrix is generally in random band or block-diagonal form with the elements representing circuit voltages, impedances or power sources [36]; in structural engineering, the sparse matrix is generally of band or block form, with the bandwidth or block dimension representing the number of joints per floor [13, 19]; in reactor diffusion problems and differential equations, the band form of matrix is most common, with the bandwidth being the number of points used in a point-difference formula [23, 24, 37].

e.g., If we solve the boundary-value differential equation $2+3t^2 = y+y'+y''$ by using a central difference approximation (3 points), and 10 intervals between the points $y(t=0) = 0$, and $y(t=1) = 1$; the resulting 9x9 matrix obtained is shown on page 6. It is a Band Diagonal matrix.

A 5-point interpolation would yield a bandwidth of 5; 50 intervals would result in a 49x49 matrix.

In favour of Band Diagonal matrices, it can be said that in most cases, they are processed either wholly by rows or columns, and little or no processing of single element occurs. Hence at any time, only a few rows/columns need be maintained in fast core for immediate use.

$$
\begin{bmatrix}
-199 & 100.5 & & & & & & & \\
99 & -199 & 101 & & & & & & \\
 & 98.5 & -199 & 101.5 & & & & & \\
 & & 98 & -199 & 102 & & & & \\
 & & & 97.5 & -199 & 102.5 & & & \\
 & & & & 97 & -199 & 103 & & \\
 & & & & & 96.5 & -199 & 103.5 & \\
 & & & & & & 96 & -199 & 104 \\
 & & & & & & & 95.5 & -199
\end{bmatrix}
$$

Therefore, it is advisable that whenever a user encounters such a matrix, a special effort is desired on his part to adapt his processing algorithms to the case at hand. This adaptation should be made because of the inherent simplicity of processing, manipulating and solving band matrices, and also because of the opportunity to minimize core allocation and execution time.

As already seen, most of the structured matrices that one encounters in different application areas are sparse, too. This is especially true in the case of large matrices because of the fact that the matrix density (defined as the number of nonzero elements of the matrix divided by the total number of elements in the full matrix) decreases as matrix size increases.

Now, if number of 'zero' entries in the structured matrix, within a band or a block as the case may be, is quite large, then it is not advantageous to store the matrix using its structural property. It pays to treat such matrices as sparse matrices only, and use the existing indexing techniques for storing sparse matrices.

To make this report self-explanatory and self-sufficient to the reader, a brief account of existing indexing techniques to store sparse matrices is included here.

The different techniques are explained below [21].

I. Bit-Map Scheme:-

The given matrix is first converted to a Boolean matrix by putting nonzero entries as 1's. These elements are stored in an array. A row-index vector is created to give address of the first nonzero entry of that row.

e.g.

$$A = \begin{bmatrix} 5 & 0 & 3 & 0 \\ 0 & 2 & 0 & 7 \\ 4 & 0 & 0 & 6 \\ 0 & 0 & 8 & 0 \end{bmatrix} \qquad BM = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$Z = (5, 3, 2, 7, 4, 6, 8)$

	Row-Index Vector		Actual Values
V(1) →	1	→ Z(1)	5 3
V(2) →	3		2 7
V(3) →	5		4 6
V(4) →	7		8

The rows of bit map are packed to store maximum number of rows in a word.

e.g.,

Byte 1	Byte 2	Byte 3
1 0 1 0 0 1 0 1	1 0 0 1 0 0 1 0	- - -

If I and J are dimensions of the matrix, and B is the number of bits /word, then this scheme requires (I*J)/B = W words. Storage required for row-index vector = I*A/B words, where A = number of bits required for an address.

∴ Percentage storage required of the full matrix

$$= \frac{100}{B}\left(1 + \frac{A}{J}\right) \% .$$

Using B = 32 bits and A = 16 bits for a 500 x 500 matrix, 5% dense; total storage required is 12,500 (for nonzero elements) + 8313 (for bit-map and row-index vector) = 20813 words = 8.325% of the full matrix.

If we wish to access a_{ij}, we note that this bit lies in $S_i = \{(i-1) * J + j + (B-1)\} / B$ word of the bit map. The required bit can now be isolated by masking all other bits by a logical operation or shifting the word.

If we store the bit-map by rows, then to perform a column operation, we shall have to count $j^{\underline{th}}$ bit for all I rows, which is a time consuming process, especially if such operations are quite frequent.

Advantages:-

1. This scheme of indexing requires a definite core storage.
2. Row-access time offerred is very reasonable.
3. Quite core efficient if matrix density is greater than 5%. .
4. If operations to be performed are mostly row operations, then it is quite fast as regards execution time.
5. Fast Boolean operations are possible.

Disadvantages:-

1. If stored by rows, it has a very poor column access time.
2. Reordering data elements requires considerable time.
3. Makes poor use of parallel processing.
4. If matrix density is very low, the scheme does not prove to be core efficient.

II. Address-Map Scheme:-

This indexing scheme is quite similar to one already described; the difference lies in the fact that here, we store an address or an address displacement for each matrix element. Obviously, this requires C times more core to store the matrix, where C = number of bits used for an address or address displacement.

If there are less than 64 nonzero entries per row, the address displacement requires only 6 bits.

The overall percentage storage requirement of the full matrix in this scheme with the row-index vector is

$$S_{add\text{-}map} = \frac{100}{B} \left[C + \frac{A}{J} \right] \% .$$

To retrieve a_{ij} th element, it is necessary to access the $S_i = \left\{ \frac{C}{B} (i-1) * J + j \right\}$ th character or byte (assuming one byte used for an address displacement). If the entry in S_i th byte is zero, the matrix element was null; otherwise the constant of S_i th byte is added to the row-index element to give the address of the nonzero element.

Advantages:-

1. In machines where character addressing is available, the scheme makes very efficient use of parallel processors.

2. Any element of the matrix can be readily accessed.

3. As compared to bit-map scheme, it requires less execution time to reorder data elements.

4. Row and column access times exhibited are very reasonable.

Disadvantages:-

1. It requires a large fast core for indexing.
2. As compared to 'threaded-list' scheme, it requires large execution time to reorder matrix elements.

III. Delta or Displacement Indexing:-

This indexing technique is almost identical to Address-map scheme. Instead of storing absolute addresses of matrix elements, address displacements from previous one are stored. In one of the existing delta-indexing schemes, one 64-bit extended precision word contains one 16-bit index and six 8-bit displacements to the index. Hence column indices of 7 elements can be referred to by loading and processing one extended precision word.

e.g.

773	797	806	873
col.index 1	C.I.2	C.I.3	C.I.4

Blocked Index Word

The corresponding delta-indexed word is

773	24	9	67	-	-	-	-
C.I.1	delta		delta		delta		delta

The two indexed words are equivalent.

IV. Threaded-List Scheme:-

This indexing method concentrates only on the nonzero elements of the matrix. For each of these, an element of an array in core is provided. Each such element consists of at least 3 components; one containing row and column indices of the element, second containing its value and the third pointing to the next array element.

e.g.,

Address	row	col.	value	pointer
773	7	31	-	809
809	27	3	-	976
883	33	25	-	983
976	41	19	-	883
983	- - -			

Minimum storage required for indexing in threaded-list scheme is

$$W = \frac{I}{B} \left[(J * T + A) D + V \right] \text{ words;}$$

where I, J are dimensions of the matrix

A = Number of bits required for an address.

B = Number of bits/word.

T = Number of bits used for column index element.

V = Number of bits used for row index element.

D = Density of the matrix.

Advantages:-

1. This is the sole technique that allows a simple and fast execution method of reordering; adding or annihilating data elements.

Disadvantages:-

1. It requires large core, especially if the matrix density is high,

2. Slower access time for columns as compared with that in 'row-column' scheme.

V. Row-Column Scheme:-

There exist plenty of row-column schemes differing from one another in construction of different arrays [31].

Scheme 1:

To each nonzero element of the matrix, there corresponds an item of two storage cells; the first containing the row-index, and the second, the value of the element. If first element is '0', we interpret it as end of the current column, in which case the second element indicates the index of the next column. Zeros in both cells indicates end of the matrix storage.

If n = dimension of the matrix, β = no. of nonzero entries, then storage required = $2(n + \beta + 1)$ cells.

e.g.,

$$A = \begin{bmatrix} 5 & 0 & 3 & 0 \\ 0 & 2 & 0 & 7 \\ 4 & 0 & 0 & 6 \\ 0 & 0 & 8 & 0 \end{bmatrix}$$

For A, storage goes as follows.

(0,1; 1,5; 3,4; 0,2; 2,2; 0,3; 1,3; 4,8; 0,4; 2,7; 3,6; 0,0).

Scheme 2:

Three arrays, namely, VE (Value of elements), RI (Row indices) and CIP (Column index pointer) are created. RI(α) contains row-index of the corresponding element VE(α) of VE. CIP indicates position of first element of a column in VE array.

Total storage required is clearly $2\beta + n$ cells.

For A,

$VE = (5, 4, 2, 3, 8, 7, 6)$

$RI = (1, 3, 2, 1, 4, 2, 3)$

$CIP = (1, 3, 4, 6)$

Scheme 3:

In this scheme, with each nonzero element of the given matrix, a unique integer θ_{ij} is associated.

$$\theta_{ij} = i + (j - 1)\ n.$$

Two arrays VE and TH (Theta) are created and stored.

For A,

$VE = (5, 4, 2, 3, 8, 7, 6)$

$TH = (1, 3, 6, 9, 12, 14, 15)$

To recover the original matrix; j is the least integer $> \theta(i,j)/n$ and $i = \theta(i,j) - (j-1)\ n$.

To store a matrix in this case, we require only 2β cells.

Scheme 4:

This indexing scheme can be used if the matrix is symmetrical. In that case, only the lower triangular part of the matrix is stored. The storage consists of two arrays, VE and PD (Position of Diagonal elements in VE). For each row, the leftmost nonzero element and all other elements to its right upto and including the diagonal element are stored in VE.

Thus the $i^{\underline{th}}$ row of the matrix needs (θ_i+1) locations, and VE will have $\sum_{i=1}^{n} (\theta_i+1)$ elements; where θ_i = distance of leftmost nonzero element from its diagonal, in $i^{\underline{th}}$ row.

To store the matrix then, requires $\sum_{i=1}^{n} \theta_i + 2n$ locations.

e.g.,

$$A = \begin{matrix} 4 & 0 & 3 & 0 \\ 0 & 7 & 0 & 2 \\ 3 & 0 & 5 & 0 \\ 0 & 2 & 0 & 1 \end{matrix}$$

Then,

VE = (4, 7, 3, 0, 5, 2, 0, 1)

PD = (1, 2, 5, 8)

An element a_{ij} can be recovered as follows.

PD(i) - (i-j) is the position of a_{ij} in VE provided PD(i) - (i-j) > PD (i-1); i.e. a_{ij} does not lie to the left of the first nonzero element of the $i^{\underline{th}}$ row, otherwise, $a_{ij} = 0$.

e.g., To recover a_{43} from VE, we have,

$$PD(4) - (4-3) = 8-1 = 7 > 5 = PD(3)$$

$\therefore$ a_{43} is in VE(7).

The discussion of different indexing techniques indicates that an exact comparison of execution times must reflect the type of mathematical manipulation being performed on the

sparse matrix. e.g., The bit-map method is of particular use when the matrix is used to produce an "optimal" ordering, so the matrix inverse will not have a great increased density. In contrast, the row-column method is faster than other methods when manipulations involve one row (col.) acting on other rows (cols.).

The second important aspect of indexing scheme selection is the conservation of execution time. If arithmetic operations are to be performed on the data, primary consideration should be given to a row-column method; if Boolean arithmetic or reordering algorithms are to be performed, the bit map scheme should be considered first; and if a great number of data elements are to be reordered, created or annihilated, a threaded-list scheme deserves first consideration.

CHAPTER 2

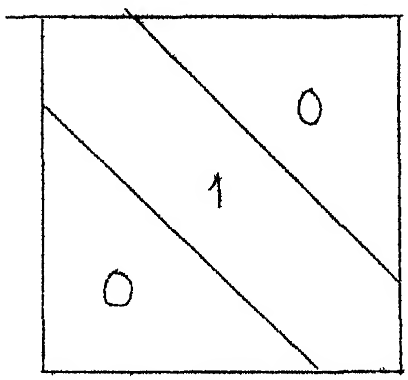
GENERATION OF STRUCTURED MATRICES

Given a random matrix, we first convert it to a Boolean matrix by putting 1 in place of each nonzero entry of the matrix. If our generation procedure is row-wise, we store an array VE (Value of Elements) which contains actual value of nonzero elements appearing in row-wise order.

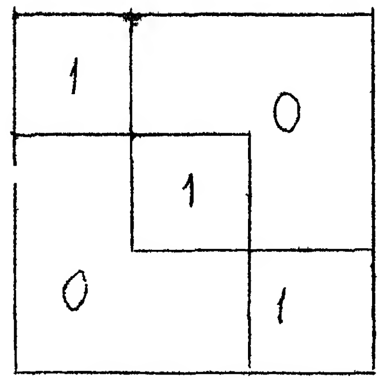
When we wish to perform operation on $i^{\underline{th}}$ row elements; we successively generate first, second, ..., $(i-1)^{\underline{th}}$ row keeping count of total number of nonzero elements occurring in those rows. Let K be this number. Then we generate $i^{\underline{th}}$ row and obtain actual values of nonzero entries from VE(K+1) onwards.

Thus, extracting exact values of matrix elements from Boolean matrix representation can be done simply by keeping a count of total nonzero entries in previous rows and then retrieving elements from the array, VE. Keeping this in mind, henceforth, we shall be confining our attention to Boolean matrices only.

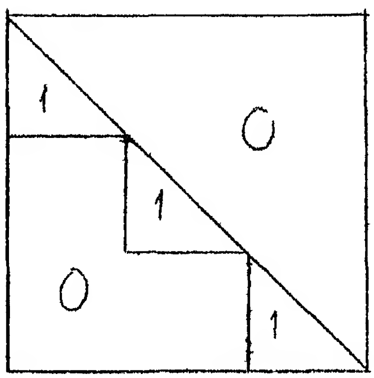
Figure 1 shows a few "structured-matrices".



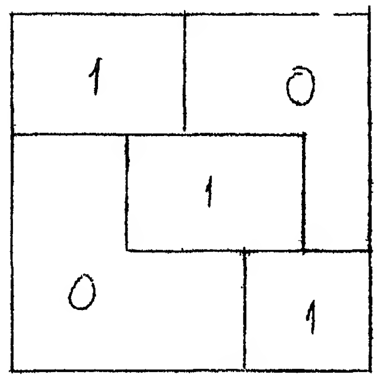
Band - Diagonal



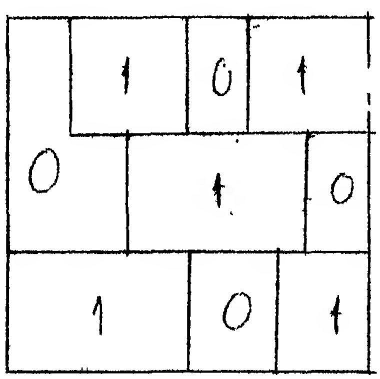
Block Diagonal



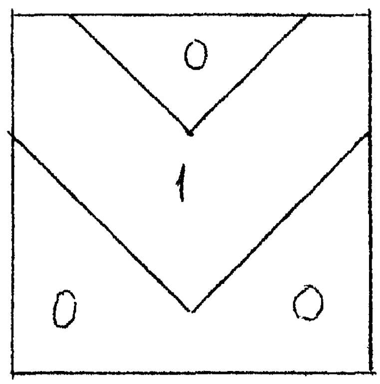
Band - Triangular



"Overlapping - Blocks"



Multiple & 'Overlapping - Blocks"



V - shaped

FIGURE - 1

It can be argued that not all of these occur in application-areas. However, as we shall consider later (Chapter 6), the combination of two of these matrices with AND, OR or EXCLUSIVE OR element-wise operation between them yield seemingly random matrix. This leads us to storing two structured matrices instead of a random matrix, which in most of the cases involving large matrices, results in reducing core-storage requirement.

The generation of these structured matrices uses shift-register technique; $(i+1)^{st}$ row being generated from the i^{th} row using the structural property of the matrix. The generating scheme is shown in Fig.2.

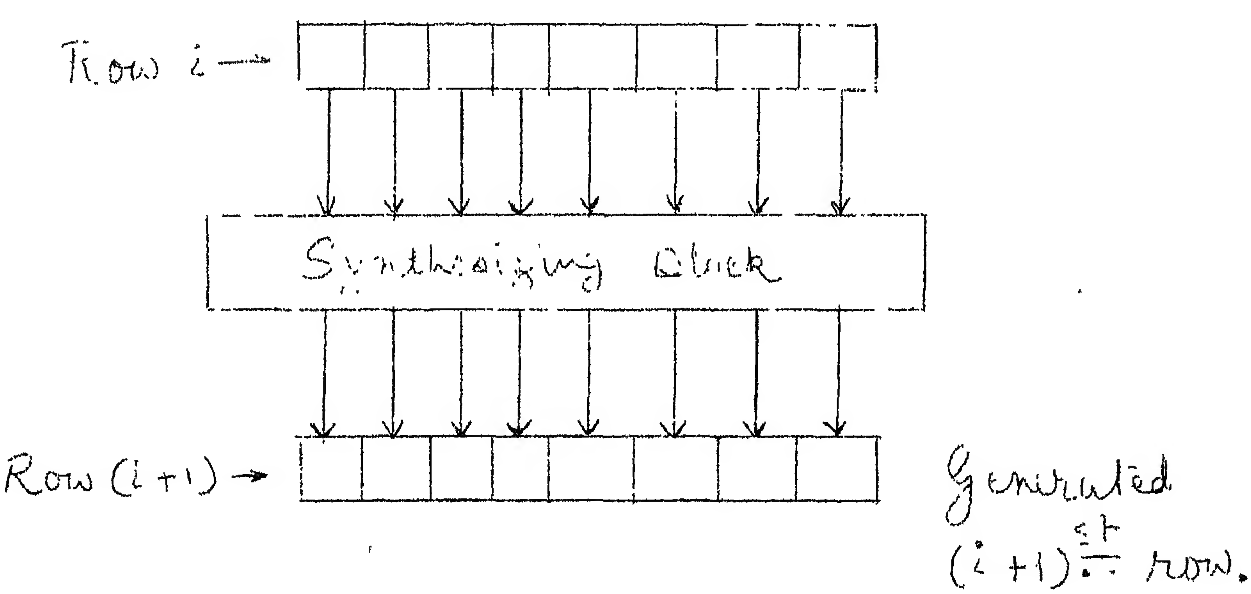


FIGURE 2

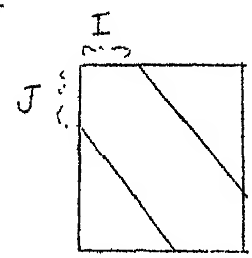
The operations allowed are as follows:-

1) Shift right
2) Shift left
3) Count and
4) Compare.

The importance of working with structured matrices is quite evident. As these can be generated row-by-row by using shift registers, it is not necessary to store the complete matrix in core at any time. Only the pertinent row can be generated and used for further computations. Clearly, this means an increase in Execution time, only as far as generation of consecutive rows is concerned. However, a problem which could not be solved earlier due to prohibitive demands on memory requirements, can now be taken up quite conveniently.

If a given sparse Boolean matrix is one of the structured-matrices, or can be converted to it by row-column permutations, (algorithms for a few random matrices are given in this and the next chapters); then the total storage required is proportional to N, the size of the matrix, while if full matrix is stored, the storage required is of the order of N^2, which is considerably large, especially in cases when N is quite large.

ALGORITHMS FOR GENERATION OF STRUCTURED MATRICES USING SHIFT REGISTER TECHNIQUES



(i) Band-Diagonal Matrix:*

Step 1: Given I,J,N. Initialize a count K=1.

Step 2: Set the shift register contents to I 1's and (N-I) 0's; i.e., the first row.

Step 3: PRINT the shift register contents.

Step 4: If all rows over, GO TO step 9.

Step 5: Increment K by 1 and shift right the contents of the shift register.

Step 6: If K > J, GO TO Step 8.

Step 7: Append a "1" in first position of shift register. GO TO Step 3.

Step 8: Append a "0" in first position of shift register. GO TO Step 3.

Step 9: STOP.

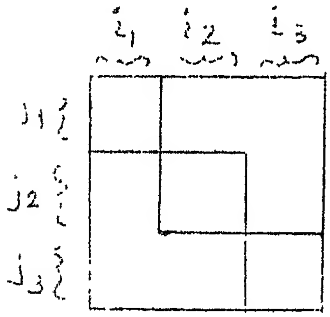
e.g., I = 2, J = 3, N = 6.

$$\begin{array}{l} \text{First row is} \\ \text{The next two rows are} \\ \\ \text{Then, the remaining} \\ \text{rows have "0" in first} \\ \text{bit position} \end{array} \quad \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

---

*Read "Band Diagonal Matrix" as "fully-packed Band Diagonal Matrix" if Bands are all 1's.

The matrix generated is the desired Band-Diagonal matrix.



(ii) Block-Diagonal Matrix

Step 1: Given the size of the matrix N and block sizes.

Step 2: Set the shift register contents to first row of the first block. PRINT it.

Step 3: If all rows over, GO TO Step 9.

Step 4: If the present block generation is complete, GO TO Step 6.

Step 5: PRINT the shift register contents. Keep the count of number of rows generated of the present block. GO TO Step 3.

Step 6: If all "blocks" generated, GO TO Step 8.

Step 7: Set the shift register contents to first row of the next block. PRINT the shift register contents and GO TO Step 3.

Step 8: Generate the remaining "ZERO" rows, if required.

Step 9: STOP.

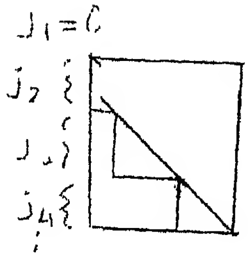
e.g., $i_1 = 2$. $i_2 = 1$. $i_3 = 2$. N = 7.

$j_1 = 3$. $j_2 = 2$. $j_3 = 1$.

It row of the first
tk is
Te such rows form first
bc
N block is generated
ao on.

$$\begin{bmatrix} 1 & 1 & 0 & \bullet & 0 & 0 & 0 \\ & & & & & & \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(: <u>Band-Triangular Matrix:</u>



St1: Given triangle sizes and dimension of the matrix.

St: Set the shift register contents, a single 1 in first bit position and all 0's, e., the first row of the first triangle.

St: PRINT the shift register content

St: If all rows over, GO TO Step 10.

St: If all rows of the present triang over, GO TO Step 7.

St: Shift right the shift register conts, and put a "1" in the column in which thewas first "1" in the last shift register conte. GO TO Step 3.

Ste If all triangles generated, GO Ttep 9.

Ste Set the shift register contents first row of the next triangle, and GO TO Ste.

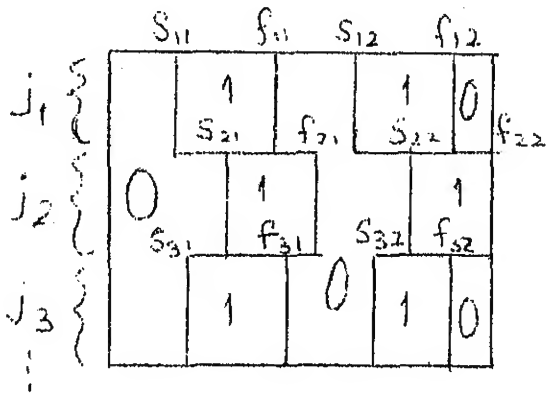
Step 9: Generate the remaining "ZERO" rows, if required.

Step 10: STOP.

e.g., $J(1) = 0$ $J(2) = 3$ $J(3) = 2$ $J(4) = 2$ $N = 7$

First row is	1 0 0 0 0 0 0
Then the first triangle is	1 1 0 0 0 0 0
generated	1 1 1 0 0 0 0
Then the next	0 0 0 1 0 0 0
triangle is gene-	0 0 0 1 1 0 0
rated and so on.	0 0 0 0 0 1 0
	0 0 0 0 0 1 1

(iv) <u>Multiple and Overlapping-blocks Matrix</u>:



Step 1: Given number of sets of blocks, starting and finishing column of each block, and size of the matrix.

Step 2: Set the shift register contents to first row using "starting and finishing columns of blocks" information.

Step 3: PRINT the shift register contents.

Step 4: If all rows over, GO TO Step 10.

Step 5: If the present set of blocks over, GO TO Step 7.

Step 6: After keeping count of the number of rows of the present set of blocks generated, GO TO Step 3.

Step 7: If all sets of blocks over, GO TO Step 9.

Step 8: Set the shift register to first row of next set of blocks, again by using "starting and finishing columns of each block" information. GO TO Step 3.

Step 9: Generate the remaining "ZERO" rows, if required.

Step 10: STOP.

e.g., $s_{11} = 1 \quad f_{11} = 3 \quad s_{12} = 5 \quad f_{12} = 6$

$s_{21} = 2 \quad f_{21} = 4$

$s_{31} = 2 \quad f_{31} = 4 \quad s_{32} = 6 \quad f_{32} = 7$

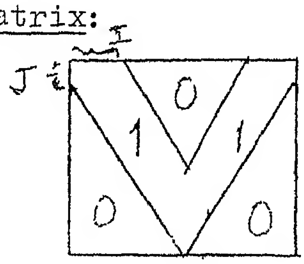
$j_1 = j_2 = j_3 = 2. \qquad N = 7.$

Then, the matrix generated is

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(v) "<u>Overlapping-blocks" Matrix</u>: Same as (iv)

(vi) <u>V-Shaped Matrix</u>:



Step 1: Given I, J, N. Initialize K = 1.

Step 2: Set SR1 to I 1's and (N-I) 0's; and SR2 to (N-I) 0's and I 1's.

Step 3: Concatenate SR1 and SR2. PRINT their contents.

Step 4: If all rows over, GO TO Step 9.

Step 5: Increment K by 1. Shift contents of SR1 right and that of SR2 left, dropping the rightmost bit of SR1 and leftmost bit of SR2.

Step 6: If K$>$J, GO TO Step 8.

Step 7: Append a "1" in first bit position of SR1 and last bit position of SR2. GO TO Step 3.

Step 8: Append a "0" in first bit position of SR1 and last bit position of SR2. GO TO Step 3.

Step 9: STOP.

e.g., With I = 1, J = 3 and N = 8; we get the matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

If the given matrix is not in one of the structured forms but is row (or column) permuted matrix of one of such forms, then it is always possible to recover the structured matrix from the given matrix by row (or column) transformations. The algorithm for doing the same are presented below.

RECOVERY OF STRUCTURED MATRICES BY ONLY ROW PERMUTATIONS OF A GIVEN MATRIX

(i) Band Diagonal Matrix:

Step 1: Given M. If the column codes of all rows are not continuous GO TO Step 21. Otherwise, calculate the row-sums and let MAXR = max. of these.

Step 2: Consider the rows having 1 in the first column. Of these row-sums, let K be the minimum.

Step 3: Find a row whose column codes are 1,2,...,K. If not found, GO TO Step 21.

Step 4: Put this row in sorted row (SR) vector. If K > MAXR, GO TO Step 6.

Step 5: Increment K by 1 and GO TO Step 3.

Step 6: If MAXR = N, GO TO Step 16.

Step 7: I = 2; J = 1.

Step 8: Find a row having column codes I, I+1, ...,(MAXR+J). If not found, GO TO Step 21.

Step 9: Put the row in SR. Increment J by 1. If all rows over GO TO Step 20.

Step 10: If (MAXR+J) > N, GO TO Step 12.

Step 11: Increment I by 1 and GO TO Step 8.

Step 12: Assign (N-MAXR+1) to K.

Step 13: Find a row having column codes K, K+1, ..., N. If not found, GO TO Step 21.

Step 14: Put this row in SR. If all rows over, GO TO Step 20.

Step 15: Increment K by 1 and GO TO Step 13.

Step 16: If there exists a row having column codes 1,2,...N; GO TO Step 18.

Step 17: K = 2. GO TO Step 13.

Step 18: Put the row in SR. If all rows over, GO TO 20.

Step 19: GO TO Step 16.

Step 20: PRINT the row-permuted matrix and STOP.

Step 21: Report that given matrix can't be converted to Band-Diagonal form, and STOP.

(ii) <u>Block Diagonal Matrix</u>:

Step 1: Given M. If column codes of all rows are not continuous, GO TO Step 7. Otherwise, calculate row-sums and put rows having ZERO row-sum at end positions of the sorted row (SR) vector. I = 1.

Step 2: Find a row having column codes as I, I+1,.... If no such row found, GO TO Step 7.

Step 3: Put this row IR in SR. If all rows over, GO TO Step 6.

Step 4: Find a row having zero distance (i.e. identical row) from IR. If such a row exists, GO TO Step 3.

Step 5: Let highest column code of IR be J. Put I = J+1 and GO TO Step 2.

Step 6: Print the permuted matrix and STOP.

Step 7: Report that matrix can't be converted to desired form and STOP.

(iii) Band Triangular Matrix:

Step 1: Given M. Place "ZERO" rows, if any, at the end positions of the sorted row vector (SR).
I = 1 J = 1.

Step 2: Find a row having column codes I, I+1, ...,J. If no such row found, GO TO Step 5.

Step 3: Put this row in SR. If all rows over, GO TO Step 8.

Step 4: Increment J by 1 and GO TO Step 2.

Step 5: If (J-I) = 0, GO TO Step 7.

Step 6: I = J and GO TO Step 2.

Step 7: Report that matrix is not convertible to desired form and STOP.

Step 8: PRINT the row permuted matrix and STOP.

(iv) "Overlapping-blocks" Matrix:

Step 1: Given M. If column codes of all rows are not continuous, GO TO Step 8. Otherwise, calculate row-sums.

Step 2: I = 1.

Step 3: Find a row having column codes I, I+1,... . In case of clash, choose the row having minimum row-sum. If no such row found, GO TO Step 8.

Step 4: Put this row in Sorted Row Vector (SR). Call it IR. If all rows over, GO TO Step 9.

Step 5: Find a row having zero distance from IR. If such a row exists, GO TO Step 4.

Step 6: Find a row which shares one or more column with IR. In case of clash, select one having maximum columns in common with IR. If clashes occur again, choose row with minimum row-sum. If such a row exists, GO TO Step 4.

Step 7: Let J be the highest column code of IR . I = J + 1 and GO TO Step 3.

Step 8: Report that matrix cannot be converted to desired form and STOP.

Step 9: PRINT the row permuted matrix and STOP.

(v) " <u>Multiple and Overlapping-blocks Matrix</u> ":

Similar to (iv).

(vi) <u>V-Shaped Matrix</u>:

Step 1: Given M. Calculate row-sums. Place "ZERO" rows in the end positions of sorted row vector (SR). N = size of the matrix. I = 1.

Step 2: Find a row having column codes I, I+1, ...I+K, N-K, ..., N with minimum K. If no such row exists, GO TO Step 14.

Step 3: Include the row in SR. If all rows over, GO TO Step 15.

Step 4: Increment K by 1. If $I+K \leq \lceil \frac{N}{2} \rceil$; GO TO Step 6.

Step 5: Set $K = \lceil \frac{N}{2} \rceil - I$.

Step 6: Find a row having column codes I, ..., I+K, N-K, ...,N. If such row exists, GO TO Step 3.

Step 7: Increment I by 1. Decrement K by 1. Set J = 1.

Step 8: If $(I+K) \leq \lceil \frac{N}{2} \rceil$, GO TO Step 11.

Step 9: If $I > \lceil \frac{N}{2} \rceil$, GO TO Step 14.

Step 10: Set $K = \lceil \frac{N}{2} \rceil - I$.

Step 11: Find row having column codes I, ..., I+K, N-K-J, ..., N-J. If no such row exists, GO TO Step 14.

Step 12: Include this row in SR. If all rows over, GO TO Step 15.

Step 13: Increment I & J by 1 and GO TO Step 8.

Step 14: Report the impossibility of converting the matrix in desired form and STOP.

Step 15: PRINT the row-permuted matrix and STOP.

Tewarson [31] has given a few more desired forms of the matrices, which are reproduced in Fig. 3.

Given the band, block and border dimensions, it is evident that these forms can be easily generated using shift register technique.

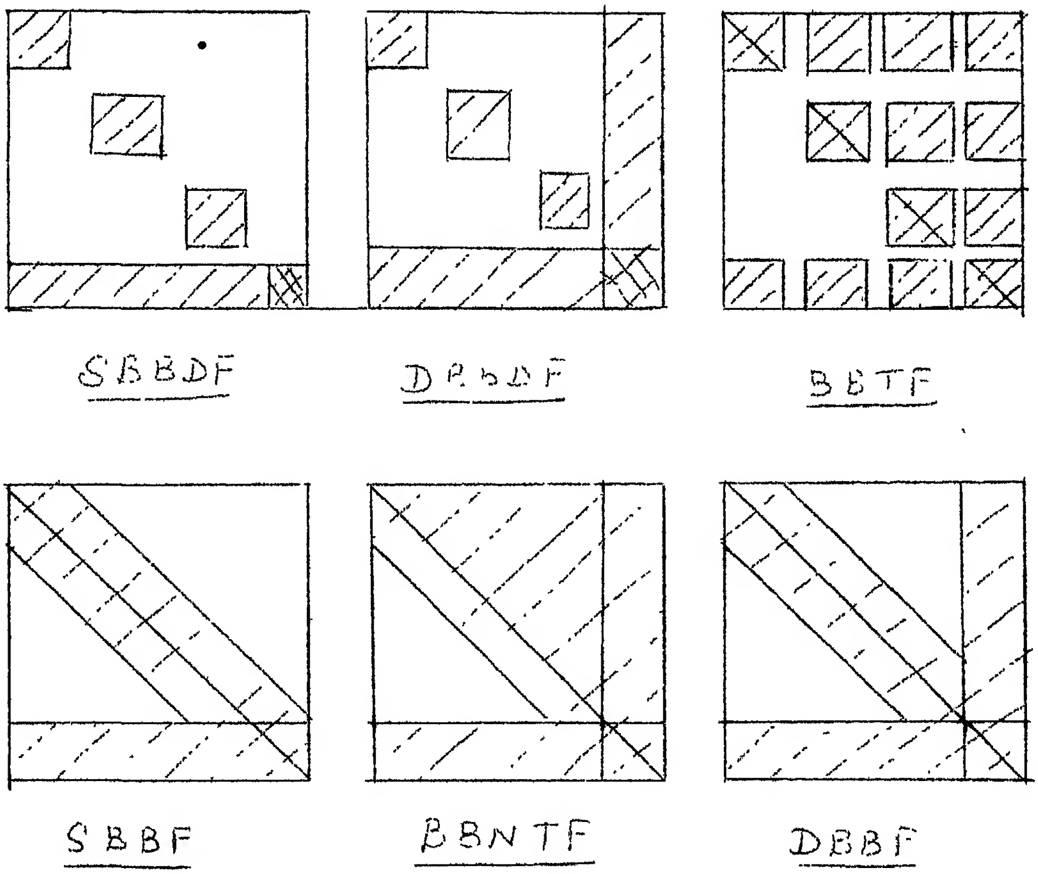


FIGURE 3

SBBDF = Singly Bordered block Diagonal form.
DBBDF = Doubly " " " "
BBTF = Bordered Block Triangular form.
BBNTF " " Band " "
SBBF = Singly Bordered Band form.
DBBF = Doubly " " "

CHAPTER 3

RECOGNITION OF STRUCTURED MATRICES

It has been seen that the generation, and recovery of structured matrices, by row-permutations only, is possible. Unfortunately, the matrices that one encounters in real-life problems do not always appear in such a concise form. In that case, the given matrix is to be converted to one of the structured forms, by employing both, row and column permutations. The algorithm for doing this are developed for three structured matrices, viz.,

i) Band Diagonal Matrix

ii) Band Triangular Matrix

iii) "Overlapping-Blocks" Matrix

Such recovery of structured matrices is of great importance. For example, when solving a large system of equations, if it is possible to reduce the coefficient matrix into a block diagonal form by row-column permutations, the blocks formed clearly indicate which equations are to be solved for finding which unknown. The technique is very well developed and explained by Steward [27,29] . Similar advantages are noticed for Band Diagonal matrix which appears quite frequently in Electrical Engineering and control problems, as well as some of the other application areas.

ALGORITHMS FOR RECOVERY OF STRUCTURED MATRICES FROM A RANDOM MATRIX, BY ROW-COLUMN PERMUTATIONS

(i) Band Diagonal Matrix:

All matrices, which can be converted to Band Diagonal form, necessarily satisfy condition C.

Condition C: (Does not apply for MAXR or MAXC = N).

If MAXR = maximum row sum, then number of rows having row-sum = MAXR should be (N-MAXR+1). Similarly, number of columns having column sum = MAXC should be (N-MAXC+1), where MAXC = maximum of column sums. The remaining rows/columns should have proper row-sums/column-sums so that these rows/columns can be placed properly. (i.e., Number of rows having row-sum = MAXR-1 should be one or two. If it is two, rows having row-sum = MAXR-2 are found, and so on. If at any stage, number of rows found = 1, then remaining rows should have row-sums = MAXR-K, MAXR-(K+1) and so on).

Step 1: Given the matrix M. Compute its row-sums and column-sums.

MAXR = maximum of row-sums.

MINR = minimum of row-sums.

MAXC = maximum of column sums.

MINC = minimum of column sums.

Step 2: If condition C not satisfied, GO TO Step 19.

Step 3: Select a row having minimum row-sums. Call it R1. Put it in sorted row vector (SR). Set IP to MINR.

Step 4: Find a row having row-sum = IP + 1, and which is not yet considered. Call it R2.

Step 5: If R2 is not found, GO TO Step 19.

Step 6: If distance between R1 and R2 $\neq$ 1, GO TO Step 4.

Step 7: Let R1 = R2. Put R1 in SR. Increment IP by 1. If IP $<$ MAXR, GO TO Step 4.

Step 8: Find a row having row-sum = MAXR and distance from R1 = 0 (in case when MAXR = N, only) or 2. If no such row found, GO TO Step 19.

Step 9: Call the row R1 and put it in SR. If all rows over, GO TO Step 14.

Step 10: If number of rows having row-sum = MAXR already considered is not greater than N-MAXR, GO TO Step 8.

Step 11: Set IP to MAXR-1.

Step 12: Select a row having row-sum = IP, which is not yet considered, and has distance 1 from R1. If no such row found, GO TO Step 19.

Step 13: Call it R1 and put it in SR. If all rows are not considered, GO TO Step 12, after decrementing IP by 1.

Step 14: If columns of M are considered, GO TO Step 16.

Step 15: Transpose M to get M'. IRD = MAXR - MINR.MAXR = MAXC etc. GO TO Step 3.

Step 16: If MINC = IRD+1, GO TO Step 18.

Step 17: Reverse the column order.

Step 18: PRINT the permuted matrix and STOP.

Step 19: Report that matrix is not convertible to Band Diagonal form and STOP.

(ii) <u>Band Triangular Matrix</u>:

Step 1: Given the matrix M. Calculate its row-sums.

Step 2: Put rows having "ZERO" row-sum at the end of sorted row vector (SR). Set INDEX = 0.

Step 3: Find a row whose row-sum = 1 and which is not yet considered. If no such row exists, GO TO Step 12.

Step 4: Find the column having non-zero entry. Call it J. If the column is already considered, GO TO Step 3.

Step 5: Increment INDEX by 1. Put selected row IR in SR vector. Interchange columns J and INDEX.

Step 6: If all rows over, GO TO Step 11.

Step 7: Set K = 2.

Step 8: Find a row having row-sum = K and which is not yet considered. If no such row found, GO TO Step 3.

Step 9: Call new found row, I. If distance between rows I and IR is not equal to 1, GO TO Step 8.

Step 10: Find the column J such that M(I,J) = 1 but M(IR,J)$\neq$1. Increment INDEX by 1 and interchange columns J and INDEX. Put $I^{\underline{th}}$ row in SR. Let IR = I and K = K+1. If all rows not over, GO TO Step 8.

Step 11: Columns are alrady permuted by interchanges. Permute rows as given by sorted row vector and PRINT the permuted matrix. STOP

Step 12: Report that matrix is not convertible to Band Triangular form and STOP.

(iii) "Overlapping-Blocks" Matrix:

Step 1: Given the matrix M. Calculate row-sums and column-sums.

Step 2: Form blocks of rows and columns and let BLR(I,J) and BLC(I,K) give rows and columns respectively which form $I^{\underline{th}}$ block.

Step 3: For each block, calculate the number of blocks which have at least one column in common with it. Let KT(I) give this information.

Step 4: If for any of the blocks, KT(I) > 2, GO TO Step 10.

Step 5: Choose a block whose KT(.) = 0 or 1 i.e. independent block or sharing at least one column with one block only. If no such block found, GO TO Step 10.

Step 6: Call the block IB and put it in sorted block vector (SB). If all blocks over, GO TO Step 9.

Step 7: Find a block which has at least a column in common with block IB. If no such block found, GO TO Step 5.

Step 8: Put the new block IB in sorted block vector. If all blocks not over, GO TO Step 7.

Step 9: SB vector gives blocks in order in which they would appear in the final matrix. Permute columns so that shared columns between blocks come at proper places, thus forming sorted column (SC) vector, Form sorted row (SR) vector of sorted rows from BLR. Add "ZERO" rows and/or columns if necessary, and print the permuted matrix. STOP.

Step 10: Report that matrix is not convertible to desired form and STOP.

CHAPTER 4

ALGORITHM FOR REDUCTION OF A GIVEN MATRIX TO BAND DIAGONAL FORM

In the previous chapter, we presented algorithms to convert a random matrix to one of three structured forms by row and column permutations. However, it is obvious that we cannot always reduce a given random matrix to one of the structured matrix forms. This can be mainly due to two reasons.

i) Our constraint that all elements of the matrix within a block or a band, as the case may be, are nonzero.

ii) The matrix itself is random.

So now, we relax our condition and say that the elements in a band or a block can be either 0 or 1, but the elements outside it are 0's. Even with the constraint relaxed, it is quite possible that a matrix cannot be converted to desired structured form. In that case, we have to remove a few rows and columns, place them as "borders", and convert the remaining matrix to one of the desired forms.

Rajaraman and Muthukrishnan[22] and Weil and Kettler[39,40] have developed algorithms to convert a random matrix to block diagonal form by possible removal of a few rows which form a border. (figure 1).

A new algorithm is developed to convert a random matrix in either a Band Diagonal form (if possible), or a

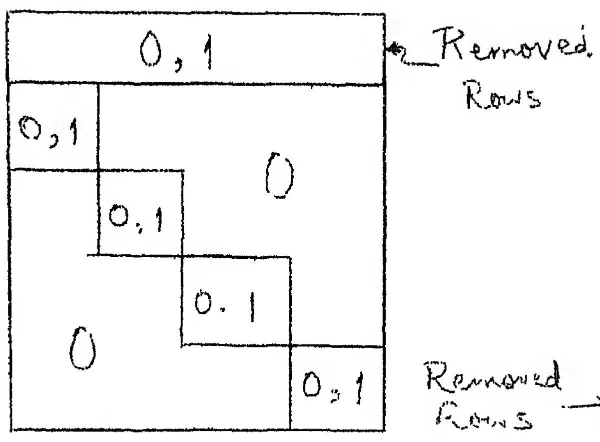


FIGURE. 1

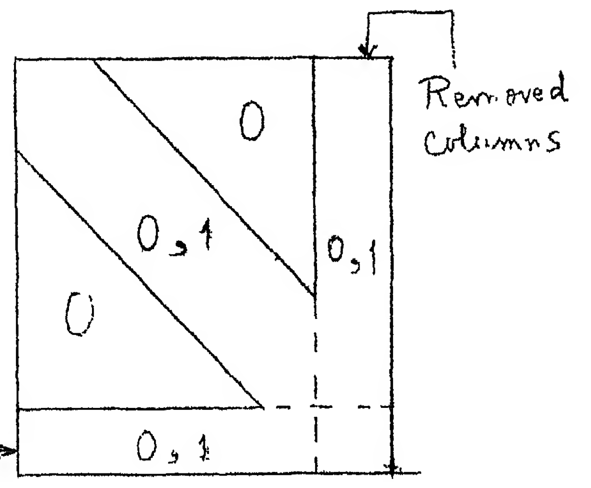


FIGURE - 2

singly or doubly bordered band form (figure 2); the latter requiring removal of rows or/and columns to reduce the band-width of the matrix.

As the nonzero elements of the matrix are now concentrated in a narrow band centered along the principal diagonal, the representation of the matrix on a computer or on paper is concise and the arithmetic entailed in the computations with it is considerably simplified.

In the algorithm, we have limited bandwidth to the maximum of row-sums and column-sums. If it increases this number, rows or/and columns are removed to decrease it. The following definitions and criterion will help understand the algorithm better.

Definitions:

A row/column is said to be adjacent to another row/column if there exists a column/row in which both of them have a nonzero entry.

If i and j are adjacent rows; and $i < j$,

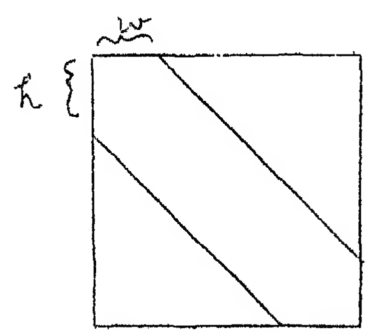
then $B(i,j) = 1$.

If k and l are adjacent columns; and $k < l$,

then $B(l,k) = 1$.

i.e., the upper triangular portion of matrix B gives row-adjacencies, while the lower triangular portion gives column adjacencies.

The Bandwidth of a band-diagonal matrix is defined to be (height of the band + width of the band -1).



$BW = h + w - 1$

In the algorithm, we first permute rows to get a new row-sequence which tends to decrease the Bandwidth. Then we consider the transposed matrix and try to find the new column sequence of the original matrix, after determining which column appears first. The desired column must satisfy the criterion C1.

Criterion C1:

We have with us, the new row-sequence. Let the first row in this sequence be R. Let P be the list of columns having nonzero entries in R. Let Q = P.

If a row has nonzero entry in either of the columns given in P, all other columns of that row which have nonzero entry in them are included in list Q. All rows are considered and list Q is made.

Now, consider the row-sequence in reverse order, and cancel out columns from Q if these have nonzero entry in that row. If at the end of consideration of one row, we have only one column in list Q, that is the desired first column. If we have empty list Q left, then we consider row-sequence from the beginning and the first nonzero entry in any of the columns of original Q determines the first column in new column sequence.

If, after getting the new permuted matrix, Bandwidth is found to be larger than the maximum number of nonzero entries in either a row or a column, then we remove rows/ columns.

We begin by removing one row first, and then one column, if necessary; and so on alternatively. When a column is to be removed, we consider the transpose matrix and remove a row, which is the same as removing a column of the original matrix.

If a row (R) is to be removed to reduce the BW of the matrix, it should satisfy criterion C2.

Criterion C2:

The element of R at the right or left end of the band should be nonzero. All elements diagonally above the element of R at the left end of the band should be zeros. So also, should the elements, diagonally below the element of R at the right end of the band.

MAIN ROUTINE

Step 1: Given the random matrix A.

Step 2: Calculate row and column sums of A. MAXM = maximum of these row and column sums. IFF = 0.

Step 3: Determine row and column adjacencies, and fill the entries of matrix B, properly.

Step 4: Call Routine P.

Step 5: The new-row sequence is obtained. Choose a column IR according to criterion C1.

Step 6: Transpose the matrices A and B.

$A \leftarrow A'$

$B \leftarrow B'$.

GO TO Routine P.

Step 7: The new column sequence is also obtained. Permute rows and columns of original A as given by two new sequences. PRINT the reduced Bandwidth matrix.

Step 8: BW = Bandwidth of the new matrix. If $BW \leq MAXM$, GO TO Step 14.

Step 9: If IFF is even, GO TO Step 11.

Step 10: A = A'.

Step 11: Check rows of A to find out which row satisfies criterion C2, so that it can be removed.

Step 12: Retranspose the matrix A, if it was transposed in Step 10.

Step 13: Make the removed row or column of A as "ZEPO" row or column. Call the new matrix A. Increment IFF by 1 and GO TO Step 2.

Step 14: PRINT the final A matrix. STOP.

ROUTINE P

Note: Whenever columns are being considered for resequencing, we are operating on A = A' (original), so treat sorted row (SR) vector as sorted column (SC) vector in that case.

Step 1: Matrix A is available.

Step 2: Choose a row which has least number of rows adjacent to it. If there is only one such row, i.e. no clashes occur, GO TO Step 4.

Step 3: Choose the first of the clashing rows whose row-sum is minimum.

Step 4: Call the chosen row IR.

If degree (IR) $\neq$ 0, i.e. at least one row is adjacent to it, GO TO Step 6.

Step 5: Put IR at the first unoccupied position from the end, of SR vector. GO TO Step 2.

Step 6: If we are considering columns of original matrix, GO TO Step 9.

Step 7: Put IR in SR VECTOR. If all rows over, GO TO Step 24.

Step 8: GO TO Step 10.

Step 9: First column is already known; Call it IR.

Step 10: Modify the incidence matrix B for row IR. If I is incident to IR, put B(I,IR) or B(IR,I) = 0 depending on I$<$IR or I$>$IR.

Step 11: Find out neighbouring rows of IR and put them in NGHBR vector.

Step 12: If NGHBR(1)$\neq$0 i.e. at least one neighbour exists, GO TO Step14.

Step 13: Call the node following IR in SR-vector as IR. GO TO Step 10.

Step 14: Of the neighbours, select that row whose remaining degree (given by modified B matrix) is minimum. Let IR be that row. If there are no clashes, GO TO Step 20.

Step 15: If we are considering columns, GO TO Step 18.

Step 16: Choose row having maximum intersection with highest labelled node so far. If there are no clashes involved, GO TO Step 20.

Step 17: Of the clashing rows, find EXCLUSIVE ORS with previous labelled rows in reverse order. Whenever a minimum EX.OR is found, call that row IR. If no such minimum is found, call the first of the clashing rows IR. GO TO Step 20.

Step 18: Of the clashing columns, find out that column whose row-code appears first in <u>already sorted-row</u> "SR" vector. Call it IR. If there are no clashes, GO TO Step 20.

Step 19: Of the clashing columns, find out one which has maximum row-code distance in <u>already sorted-row</u> "SR" vector.

Step 20: Put IR in SR-Vector. If all rows over GO TO Step 24.

Step 21: Compress NGHBR vector removing IR from it. If there are no rows in NGHBR vector, GO TO Step 23.

Step 22: Modify adjacency matrix B in so much that adjacency of IR with other rows presented in NGHBR vector only is affected. GO TO Step 12.

Step 23: Let DEGREE (IR) = 0. GO TO Step 13.

Step 24: RETURN to Main Routine.

<u>Example</u>:

Say, we are given a set of linear equations.

$$a_{11}\, x_7 + a_{12}\, x_9 = 0$$

$$a_{21}\, x_1 + a_{22}\, x_5 = 0$$
$$a_{31}\, x_3 + a_{32}\, x_6 = 0$$
$$a_{41}\, x_5 + a_{42}\, x_8 = 0$$

$$a_{51}\, x_2 + a_{52}\, x_3 + a_{53}\, x_6 = 0$$

$$a_{61}\, x_4 + a_{62}\, x_9 = 0$$

$$a_{71}\, x_7 + a_{72}\, x_9 = 0$$

$$a_{81}\, x_1 + a_{82}\, x_5 + a_{83}\, x_8 = 0$$

$$a_{91}\, x_2 + a_{92}\, x_6 = 0$$

The coefficient matrix A is . . . (9 x 9)

$$\begin{array}{cc}
 & \begin{matrix} A & B & C & D & E & F & G & H & I \end{matrix} \\
\begin{matrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \\ i \end{matrix} &
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0
\end{bmatrix}
\end{array}$$

When converted to a Block Diagonal matrix by row and column permutations, A becomes

$$\begin{array}{cc}
 & \begin{matrix} H & A & E & I & G & D & B & F & C \end{matrix} \\
\begin{matrix} h \\ d \\ b \\ f \\ g \\ a \\ i \\ c \\ e \end{matrix} &
\begin{bmatrix}
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1
\end{bmatrix}
\end{array}$$

Looking at the coefficient matrix now, one can easily solve the given equations.

CHAPTER 5

GENERATABLE CLASSES OF MATRICES

We have used the shift register technique for generation of structured matrices. For convenience's sake, we redraw the figure here.

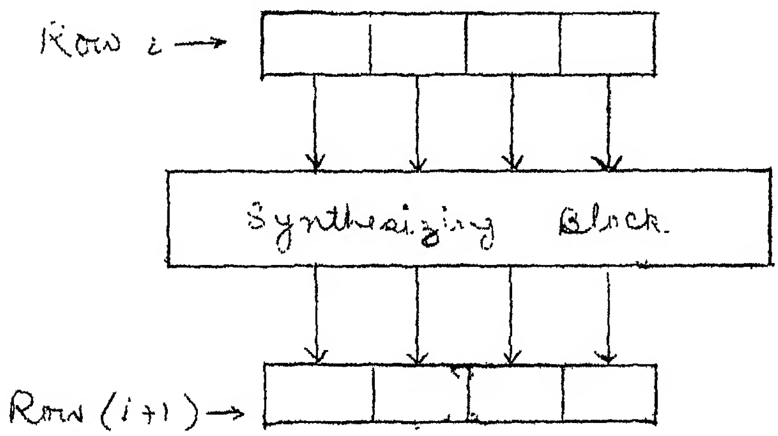


FIGURE - 1

The $(i+1)^{\underline{st}}$ row of the matrix is obtained from $i^{\underline{th}}$ row using the structural properties of the matrix. The allowed operations are ...

i) Shift left;

ii) Shift right;

iii) Count; and

iv) Compare.

An attempt is made to characterize the classes of matrices that can be generated using shift register technique and these four operations. It is based solely on the

structural aspect of matrices, and seems to be a little vague in nature. The mathematical characterization of such classes may be possible and can be attempted in future.

Initially, assume that only operation allowed is "shift right", and a "zero" is appended at the leftmost bit position. In that case,

$$r_{i+1} = 0, r_i (1_r)$$

where, $r_{i+1} = (i+1)^{\underline{st}}$ row

, implies concatenation

$r_i(1_r) = i^{\underline{th}}$ row with a rightmost bit clipped.

The class of matrices that can be generated in this consists of ~~all~~ only those matrices in which there appear bands in upper triangular portion of the matrix, the element in lower triangular portion being zeros.

Similarly, if the operation allowed is "shift left" only, and a "zero" is appended at the rightmost bit position, then...

$$r_{i+1} = r_i(1_l), 0 \quad ,$$

where, $r_i(1_l) = i^{\underline{th}}$ row with a leftmost bit clipped.

And, all matrices having bands running from N-E to S-W; and elements in S-E portion of the matrix being zeros, fall in the class of generatable matrices.

If instead of "zero" being appended, we append a "1" at the rightmost or leftmost bit position in cases where

operation allowed is "shift left" only and "shift right" only respectively, then the two generatable classes of matrices are respectively,...

i) all matrices having bands running from N-E to S-W; with elements in S-E portion of the matrix being nonzeros; and

ii) all matrices having bands running from N-W to S-E, with elements in lower triangular portion being nonzeros.

Next we consider the case when the only operation allowed is "shift right", but now either 0 or 1 can be appended at the leftmost bit position; i.e.,

$$r_{i+1} = 0, \quad r_i(1_r) + 1, \quad r_i(1_r),$$

where + indicates OR.

In this case, all matrices having bands running from N-W to S-E fall into the class of generatable matrices.

Similarly, if we have only "shift left" operation with either 0 or 1 appended at the rightmost bit position, then all matrices having bands running from N-E to S-W fall into the class of generatable matrices.

Let us now allow two operations, viz., "Shift left" and "shift right", with "zero" appending only. Then...

$$r_{i+1} = 0, \quad r_i(1_r) + r_i(1_l), \; 0$$

e.g., $r_1 =$ 1 0 0 1 1 0 1

Shift right → 0 1 0 0 1 1 0

Shift left ← 1 0 0 1 1 0 0

← 0 0 1 1 0 0 0

→ 0 0 0 1 1 0 0

→ 0 0 0 0 1 1 0

← 0 0 0 1 1 0 0

To calculate distance between r_i and r_{i+1}; the following algorithm can be used.

Step 1: DIST = 2* No. of cluster of 1's in r_i.

Step 2: If shift right and 1_r of r_i is nonzero, GO TO Step 5.

Step 3: If shift left and 1_l of r_i is nonzero, GO TO Step 5.

Step 4: GO TO Step 6.

Step 5: DIST = DIST-1

Step 6: STOP.

This class includes all those matrices which have bands of 1's running from either N-W to S-E or N-E to S-W; but all bands are symmetrical in their run, i.e. all change direction of orientation at the same row.

In the previous case, if we have appending of 1's instead of 0's; then...

$$r_{i+1} = 1,\ r_i(1_r) + r_i(1_l),\ 1$$

To calculate the distance between i^{th} and $(i+1)^{th}$ row, same algorithm can be used by replacing the term "nonzero"

by "zero".

This class comprises of all those matrices which have bands of zeros running from either N-W to S-E or N-E to S-W; but all of them being symmetrical in their run.

Now, if along with two operation, we allow **appending** by both zero and nonzero; then....

$$r_{i+1} = 0,\ r_i(1_r) + 1,\ r_i(1_r) + r_i(1_l),\ 0 + r_i(1_l),\ 1$$

e.g.

$$\begin{array}{ll|ccccccc}
r_1 & = & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\
R - 0 & & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
L - 1 & & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\
L - 0 & & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\
L - 1 & & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\
R - 1 & & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\
R - 1 & & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\
L - 0 & & 1 & 0 & 1 & 0 & 0 & 1 & 0
\end{array}$$

R - 0 implies shift right, append a zero.

To calculate distance between r_l and r_{i+1}, the following algorithm can be used.

Step 1: If "0" appended, GO TO Step 2.

DIST = 2* No. of cluster of zeros.

If operation is R and 1_r of r_i is 0, DIST = DIST-1.

If operation is L and 1_l of r_i is 0, DIST = DIST-1.

stop.

Step 2: DIST = 2* No. of cluster of 1's.

If operation is R and 1_r of r_i is 1, DIST = DIST-1.

If operation is L and 1_l of r_i is 1, DIST = DIST-1.

STOP.

The class comprises of all those matrices which have bands of zeros and ones running symmetrically (i.e. changing direction of orientation at the same instant) down the matrix.

Let us now divert our attention to two other operations, count and compare.

It seems possible to generate any matrix having some type of "blocks" in it using these two operations. All these matrices form the class of generatable matrices using these operations. The most general block matrix is shown in figure 2.

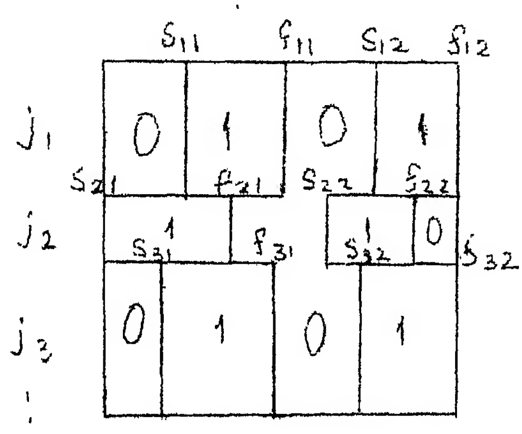


FIGURE 2

Knowing all s's, f's and j's, the matrix can be generated by keeping track of number of rows generated and changing the shift register contents when one row of blocks is over.

Now, say we allow three operations, viz., count, compare and shift right with either "0" or "1" appending. We shall also put one restriction that $(i+1)^{\underline{st}}$ row is generated by using shift right operation on $i^{\underline{th}}$ row.

It is easy to see that this class comprises of multiple-band matrices running from N-W to S-E as shown in Fig.3.

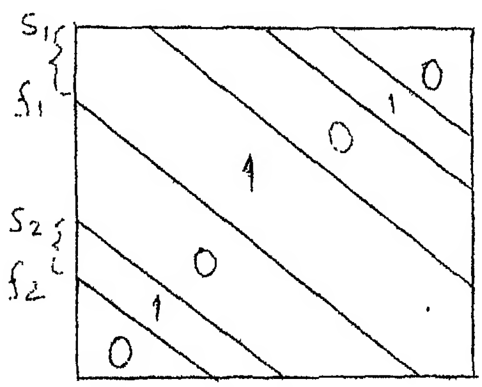


FIGURE 3

Given the first row and all s's and f's, the matrix can be generated by shifting right the $i^{\underline{th}}$ row and appending a "0" or "1" as the case may be to get $(i+1)^{\underline{st}}$ row.

Similarly, if we allow shift left with either "0" or "1" appending along with count and compare, the class of generatable matrices includes all multiple-band matrices with bands running from N-E to S-W.

If we relax the restriction that $(i+1)^{\underline{st}}$ row can be generated from $i^{\underline{th}}$ row by either shift left or shift right operation only, it becomes almost impossible to characterize the class, based on some structural property. Hence, the case is not considered here.

CHAPTER 6

REPRESENTATION OF MATRICES

In the previous chapter, we described the classes of matrices that can be generated using shift register technique. The characterization of these classes was based purely on the structural aspects of the matrix.

We now present some representations of matrices. The research on these representations was undertaken for the following reasons.

i) To characterize the classes of matrices, which can be generated using shift register technique, mathematically, so that it becomes simple to determine, to which class does a matrix belong.

ii) Given a matrix, to determine whether it is a combination of two or more of the structured matrices (i.e., obtainable from two structured matrices by AND, OR or EXCLUSIVE OR element-wise operations on them).

iii) To represent a given matrix completely and uniquely by some method. Storing the matrix in its new form may reduce the overall storage requirement of the matrix.

Literature survey revealed the following three types of representations, viz.,

i) Functional Representation;

ii) Polynomial Representation [17]; and

iii) Designation-number Representation [12].

We investigated a fourth representation in this thesis, namely,

iv) Walsh-function representation

It should be mentioned that these represenations of matrices proved to be quite ineffective, in that, it was not possible to determine to which class a matrix belongs using either of them. However, they are possible representations and future work may be carried out along these lines though it does not seem to be very promising.

<u>Functional Representation</u>:-

Let us take an example. Say, a row is

1 0 0 1 0 1 1 1 0 0 1 $\quad (n=11)$

It can be represented by $[B,r]$; where,

r = column number of last nonzero entry in the row,

and B is a function given by

$$B(r) = \begin{cases} k & \text{if entry in } k^{th} \text{ column is 1 and there are } 0\text{'s between } k^{\underline{th}} \text{ column and } r^{\underline{th}} \text{ column.} \\ -k & \text{if entry in } k^{\underline{th}} \text{ column is 1 and there are 1's between } r^{\underline{th}} \text{ and } k^{\underline{th}} \text{ columns.} \\ 0 & \text{otherwise.} \end{cases}$$

B is applied recursively to k to yield the given row.

For the above example, $r = 11$ and B is defined as

$B(11) = 8 \qquad B(8) = -6 \qquad B(6) = 4$

$B(4) = 1 \qquad$ all other B's $= 0$.

Now, we can store $\begin{bmatrix} B, r \end{bmatrix}$ instead of the complete row. We can go a step further and after assuming the maximum number of nonzero entries in a row to be $= p$, redefine the function B as

$B(1) = 8 \qquad B(2) = -6 \qquad B(3) = 4$

$B(4) = 1$.

That way, we require p storage cells for each row, and if the matrix is quite sparse, we reduce the total storage requirement quite a bit from N^2 to p.N. However, core required when we store only the nonzero entries is $\leq$ p.N for the whole matrix, and in comparison we do not gain anything.

<u>Polynomial Representation:-</u>

The polynomial representation of a matrix can be explained best by an example

$$M = \begin{matrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & C & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{matrix}$$

View the column j as x^{j-1} and write row polynomials of the matrix.

row 1: $1 + x$

2: $1 + x^2$

3: $x^2 + x^3$

4: $x^2 + x^4$

5: x^5

6: $x^4 + x^5 + x^6$

7: x^6

Combine all row-polynomials to write the matrix-polynomial as follows. Whenever a row changes, change the sign of the coefficient. Also view the $0^{\underline{th}}$ power of x in $j^{\underline{th}}$ row as (highest power of x in $(j-1)^{\underline{th}}$ row + 1).

For M, the matrix - polynomial is

$$1 + x - x^2 - x^4 + x^7 + x^8 - x^{11} - x^{13} + x^{19} - x^{24} - x^{25} - x^{26} + x^{33}$$

So, the sequence representing the matrix is

1 1 −1 0 −1 0 0 1 1 0 0 −1 0 −1 0 0 0

0 0 1 0 0 0 0 −1 −1 −1 0 0 0 0 0 0 1

The sequence represents the matrix completely and uniquely.

However, the sequence needs 34 words of computer storage which is much more than what would be required if we store only the nonzero entries. But if we are able to define row-polynomial in such a fashion that the final matrix-polynomial is "periodic", then we shall certainly reduce the

storage requirements.

It seems that this kind of representation would not help much in case of large matrices of order 50, or more.

Designation-number Representation:-

The concept of designation numbers is very well explained by Ledley [12].

Let E be the given $2^k \times 2^k$ matrix. Then, designation number of E ($\# E$) is the sequence of 0's and 1's obtained by placing all rows side-by-side.

Define 2k variables $A_1, A_2, \ldots, A_k, A_{k+1}, \ldots, A_{2k}$; the first k for 2^k rows and last k for 2^k columns.

$\# A_i = \left[(2^{i-1}\ 0\text{'s}), (2^{i-1}\ 1\text{'s}) \right]$ repeated to give a total sequence of length 2^{2k}.

Then, express $\# E$ in terms of corresponding elements in $\#$ A's. The expression of E obtained in terms of A's specifies the matrix completely and uniquely.

e.g. $$E = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$E = 1\,0\,0\,1 \quad 0\,1\,0\,1 \quad 1\,0\,0\,0 \quad 0\,0\,1\,0$$

$$A_1 = 0\,1\,0\,1 \quad 0\,1\,0\,1 \quad 0\,1\,0\,1 \quad 0\,1\,0\,1$$

$$A_2 = 0\,0\,1\,1 \quad 0\,0\,1\,1 \quad 0\,0\,1\,1 \quad 0\,0\,1\,1$$

$$A_3 = 0\,0\,0\,0 \quad 1\,1\,1\,1 \quad 0\,0\,0\,0 \quad 1\,1\,1\,1$$

$$A_4 = 0\,0\,0\,0 \quad 0\,0\,0\,0 \quad 1\,1\,1\,1 \quad 1\,1\,1\,1$$

$$E = \bar{A}_1 \bar{A}_2 \bar{A}_3 \bar{A}_4 + A_1 A_2 \bar{A}_3 \bar{A}_4 + A_1 \bar{A}_2 A_3 \bar{A}_4 + A_1 A_2 A_3 \bar{A}_4$$

$$+ \bar{A}_1 \bar{A}_2 \bar{A}_3 A_4 + \bar{A}_1 A_2 A_3 A_4$$

or $E = \bar{A}_1 \bar{A}_2 \bar{A}_3 + A_1 \bar{A}_4 A_3 + A_1 \bar{A}_4 A_2 + \bar{A}_1 A_2 A_3 A_4.$

For matrices in which the final expression of E becomes simplified, it may be worth storing it instead of the full matrix. However, it is observed that for a random matrix, the expression does not simplify to a great extent and hence this representation is of no use as far as storage requirement reduction is concerned.

Walsh-function Representation:-

A system $f(j,x)$ of real and almost everywhere nonvanishing functions is called orthogonal in the interval $x_0 \leq x \leq x_1$ if

$$\int_{x_0}^{x_1} f(j,x)\, f(k,x)\, dx = X_j\, \delta_{jk}$$

$$\delta_{jk} = 1 \quad \text{for} \quad j=k.$$

$$\delta_{jk} = 0 \quad \text{for} \quad j \neq k.$$

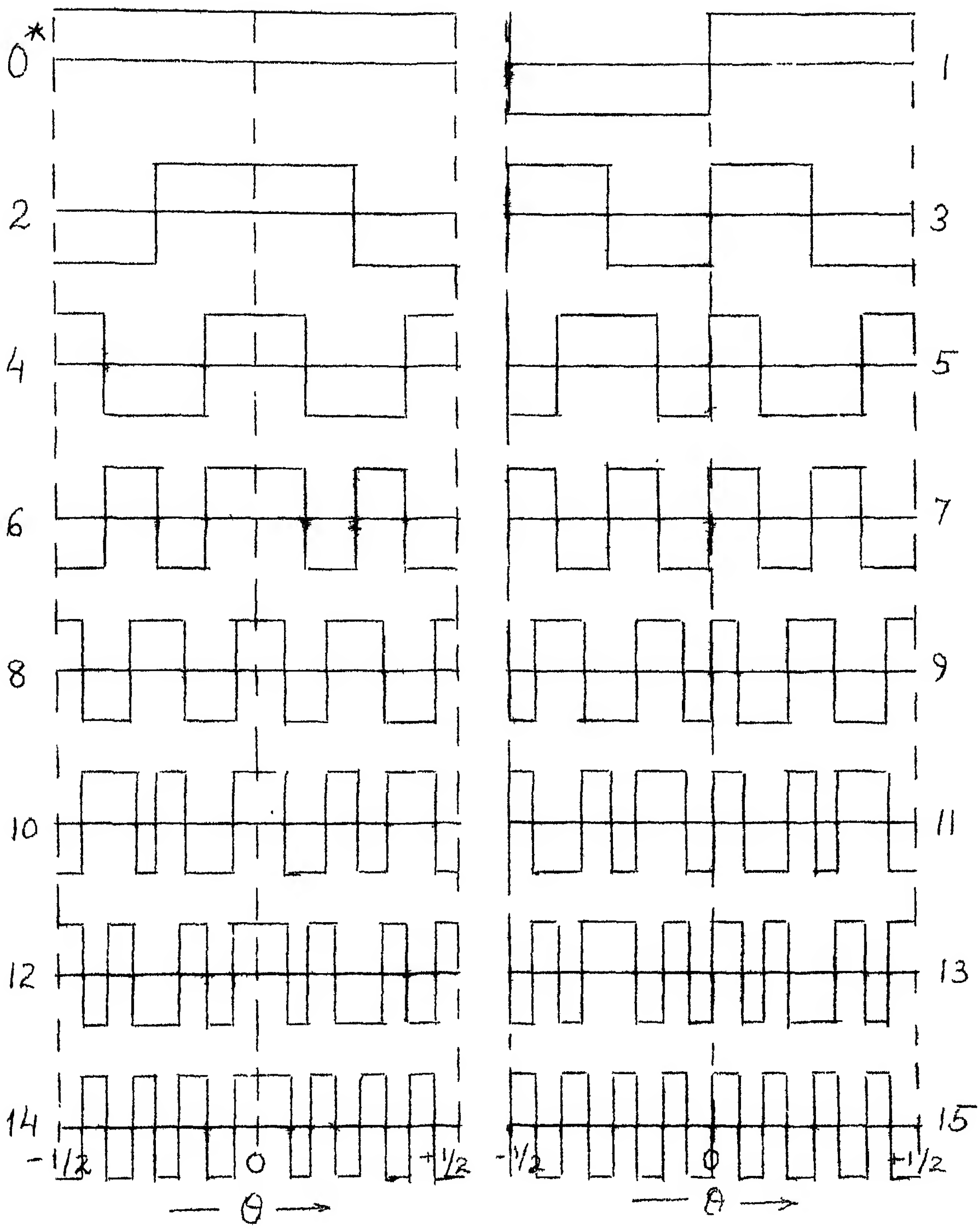


FIGURE - 1

* $0 \Rightarrow Wal(0,\theta)$

If $X_j = 1$; functions are said to be <u>orthonormal</u> [16].

One of the sets of orthonormal functions is Walsh Functions. They are shown diagrammetically in Fig. 1.

There is a close connection between 'Sal' and 'Sine' functions as well as between 'Cal' and 'Cosine' functions.

$$\text{Wal}\ (2i, \theta) = \text{Cal}\ (i, \theta); \ \text{Wal}\ (2i-1, \theta) = \text{Sal}\ (i, \theta)$$

$$i = 1,\ 2,\ 3,\ \ldots$$

The product of two Walsh functions yields another Walsh function.

$$\text{Wal}\ (h, \theta).\ \text{Wal}\ (k, \theta) = \text{Wal}\ (r_1 \theta)$$

where, $r = h \oplus k$ in binary.

e.g. $\text{Wal}\ (6, \theta).\ \text{Wal}\ (12, \theta) = \text{Wal}\ (10, \theta).$

$$\begin{array}{rl} 6 = & 0\ 1\ 1\ 0 \\ 12 = & \underline{1\ 1\ 0\ 0} \\ & 1\ 0\ 1\ 0 \ = 10 \end{array}$$

Walsh functions form a commutative group with respect to multiplication.

Plenty of such properties of Walsh and Walsh-Paley functions, and their applications are available in references [4, 10, 20, 38, 44, 45, 46, 47].

The first four Walsh elements are

$$\begin{array}{lcrrrr}
a_1 & = & 1 & 1 & 1 & 1 \\
a_2 & = & -1 & -1 & 1 & 1 \\
a_3 & = & -1 & 1 & 1 & -1 \\
a_4 & = & 1 & -1 & 1 & -1
\end{array} \quad \text{——— (1)}$$

We stick to ternary (-1,0,1) number system and view the real line as

$$\begin{array}{ccccccccccc}
\cdots & -4 & -3 & -2 & -1 & 0 & 1 & 2 & 3 & 4 & \cdots \\
\cdots & -1 & 0 & 1 & -1 & 0 & 1 & -1 & 0 & 1 & \cdots
\end{array}$$

$$\text{e.g.} \quad a_1 - a_2 = \begin{array}{rrrrr}
 & 1 & 1 & 1 & 1 \\
\ominus & -1 & -1 & 1 & 1 \\
\hline
 & -1 & -1 & 0 & 0
\end{array}$$

We also adopt a notation that if a row R = 1001; then $R = e_1 + e_4$; i.e. e's specify position of 1's in a row.

(1) gives us

$$\begin{aligned}
a_1 &= e_1 + e_2 + e_3 + e_4 \\
a_2 &= -e_1 - e_2 + e_3 + e_4 \\
a_3 &= -e_1 + e_2 + e_3 - e_4 \\
a_4 &= e_1 - e_2 + e_3 - e_4
\end{aligned}$$

Solving for e's gives ...

$$
\begin{aligned}
e_1 &= a_1 - a_2 - a_3 + a_4 \\
e_2 &= a_1 - a_2 + a_3 - a_4 \\
a_3 &= a_1 + a_2 + a_3 + a_4 \\
a_4 &= a_1 + a_2 - a_3 - a_4
\end{aligned}
\qquad (3)
$$

Now, any 4-column row can be expressed in terms of Walsh elements.

e.g. $1 \quad 0 \quad 1 \quad 1 \;=\; e_1 + e_3 + e_4$

$$= 3a_1 + a_2 - a_3 + a_4$$

$$= a_2 - a_3 + a_4$$

For first 8 Walsh elements, the Table is shown in Figure 2.

	e_1 ↓	e_2 ↓	e_3 ↓	e_4 ↓	e_5 ↓	e_6 ↓	e_7 ↓	e_8 ↓
a_1 →	1	1	1	1	1	1	1	1
a_2 →	-1	-1	-1	-1	1	1	1	1
a_3 →	-1	-1	1	1	1	1	-1	-1
a_4 →	1	1	-1	-1	1	1	-1	-1
a_5 →	1	-1	-1	1	1	-1	-1	1
a_6 →	-1	1	1	-1	1	-1	-1	1
a_7 →	-1	1	-1	1	1	-1	1	-1
a_8 →	1	-1	1	-1	1	-1	1	-1

FIGURE 2

e.g. $e_6 = a_1 + a_2 + a_3 + a_4 - a_5 - a_6 - a_7 - a_8$

$a_4 = e_1 + e_2 - e_3 - e_4 + e_5 + e_6 - e_7 - e_8$

and so on.

Now any 8 bit row can be expressed in terms of these 8 Walsh elements. However, there is one snag.

Say, we are expressing N bit numbers by N Walsh-elements, $N = 2^n$.

If $\frac{N}{3}$ leaves remainder 1; we get positive and true result, as in the case when N = 4 or 16. But if $\frac{N}{3}$ leaves remainder 2, we get negative result ($\because$ 2 on real line is -1 in our system).

e.g. for N = 8

$e_1 = a_1 - a_2 - a_3 + a_4 + a_5 - a_6 - a_7 + a_8$

$= \quad -1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0$

Hence, for representation of such rows whose $\frac{N}{3}$ leaves remainder 2, we negate the Walsh elements to correctly represent it.

e.g. 1 0 1 1 0 0 1 0 $= e_1 + e_3 + e_4 + e_7$

$= a_1 + a_2 + a_4 + a_6 - a_8$

$\therefore$ True representation is $-a_1 - a_2 - a_4 - a_6 + a_8$

Similar tables as shown in FIGURE 2 can be developed for N = 16, 32 and so on.

Let us now represent rows of a Band Diagonal matrix in terms of Walsh elements.

$$M = \begin{matrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{matrix}$$

Negate all results as $\frac{8}{3}$ leaves remainder 2.

$$R_1 = -(e_1 + e_2 + e_3) = a_3 - a_4 + a_5 - a_6 + a_7 - a_8$$
$$R_2 = -(e_1 + e_2 + e_3 + e_4) = -a_1 + a_2$$
$$R_3 = -(e_2 + e_3 + e_4 + e_5) = -a_1 - a_2 + a_3 + a_6 + a_7$$
$$R_4 = -a_1 - a_3$$
$$R_5 = -a_1 + a_2 + a_3 - a_6 + a_7$$
$$R_6 = -a_1 - a_2$$
$$R_7 = a_3 + a_4 + a_5 + a_6 + a_7 + a_8$$
$$R_8 = a_1 + a_2 - a_3 - a_4.$$

We find that even in terms of Walsh elements, there is no regularity in expression of rows. And hence, storing Walsh-element equivalents instead of rows themselves does not lead to core-allocation saving.

Matrix polynomial was discussed in Polynomial representation discussion. It consists of a series of 1's, -1's and 0's, and hence that too can be expressed in terms of Walsh elements. As earlier representation, though this is not very

helpful from storage reduction point of view, is a possible representation of matrix polynomial.

In Chapter 2, we considered 6 structured matrices, not all of which directly appear in the application areas. However, sometimes it is possible to express a seemingly random matrix in terms of two structured matrices, and core required to store two generatable matrices is less than that for a random one.

e.g.

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \wedge \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Such representation is especially useful for large matrices.

An algorithm was developed to find out the combination

of structured matrices, which would give the random matrix, but it is quite vague and hence is not mentioned here.

In this chapter, we have described our unsuccesful attempts. The work along these lines may be carried out in future but does not look promising.

CHAPTER 7

CONCLUSIONS

It is really surprising that though many authors have mentioned the occurrence of structured matrices in different fields, none of them has dealt with techniques of reducing computer storage using structural properties of such matrices. In this thesis, we have given techniques of generating, recognizing and storing structured Boolean matrices, with the intention to reduce the computer storage requirement of such matrices.

Basically, we have considered six structured matrices, and have developed methods of generating these, using shift register technique. Apart from these we have presented algorithms for . . . .

i) recognizing different structured matrices from their row or column permuted matrices,

ii) recognizing three of these structured matrices from their row and column permuted matrices, and

iii) reducing the bandwidth of a given random matrix by possible removal of rows and columns.

The generatable classes of matrices using shift register technique are characterized on the basis of their structural properties. To characterize the classes mathematically, some representations of matrices e.g. Polynomial representation, Walsh-function representation are given. However, these

proved to be our futile attempts and problem of characterizing the classes of matrices considered in this thesis, remains unsolved.

```
C===
C===   GENERATION OF BAND-DIAGONAL MATRIX
C===
C=== NOTATIONS ARE...
C===   N=SIZE OF THE MATRIX
C===   I=HORIZANTAL DIMENSION OF THE BAND
C===   J=VERTICAL DIMENSION OF THE BAND
C===
      INTEGER A(30),B(30)
      READ 101, N,I,J
      PRINT 102, N,I,J
      PRINT 103
      K=1
      DO 5 ID=1,I
 5    A(ID)=1
      IF(I.EQ.N) GO TO 7
      I1=I+1
      DO 6 ID=I1,N
 6    A(ID)=0
 7    PRINT 100,(A(II),II=1,N)
      K=K+1
      IF(K.GT.N) STOP
      DO 8 ID=1,N
 8    B(ID)=A(ID)
      IF(K.GT.J) GO TO 15
      A(1)=1
 20   CONTINUE
C===  SHIFT RIGHT OPERATION IS DONE BELOW...
      DO 9 ID=2,N
      ID1=ID-1
 9    A(ID)=B(ID1)
      GO TO 7
 15   A(1)=0
      GO TO 20
 101  FORMAT (3I2)
 102  FORMAT (1X,*THE SIZE OF THE MATRIX=*,I2/1X,*HORIZONTAL
     1 BAND DIMENSION =*,I2/1X,*VERTICAL BAMD DIMENSION=*,I2)
 103  FORMAT (/1X,* THE GENERATED MATRIX IS...*)
 100  FORMAT (/1X,30I2)
      END
£ENTRY
100304
```

THE COMPUTER OUTPUT FOR THE TEST-DATA IS...

```
THE SIZE OF THE MATRIX =10
HORIZONTAL BAND DIMENSION = 3
VERTICAL BAND DIMENSION = 4

THE GENERATED MATRIX IS...

 1 1 1 0 0 0 0 0 0 0
 1 1 1 1 0 0 0 0 0 0
```

1	1	1	1	1	0	0	0	0	0
1	1	1	1	1	1	0	0	0	0
0	1	1	1	1	1	1	0	0	0
0	0	1	1	1	1	1	1	0	0
0	0	0	1	1	1	1	1	1	0
0	0	0	0	1	1	1	1	1	[illegible]
0	0	0	0	0	1	1	1	1	[illegible]
0	0	0	0	0	0	1	1	1	[illegible]

```
C---
C---   GENERATION OF BLOCK DIAGONAL MATRIX
C---
C---   NOTATIONS ARE...
C---   N=SIZE OF THE MATRIX
C---   M=NUMBER OF BLOCKS
C---   I,J=ARRAYS SPECIFYING DIMENSIONS OF THE BLOCKS
C---
       INTEGER A(30),I(30),J(30)
       READ 101, N,M,(I(K),J(K),K=1,M)
       PRINT 102,N,M
       PRINT 103
       L=1
       LJ=1
 1     LL=0
C---   ROW CREATION DONE UPTO STATEMENT NO. 8
       DO 5 ID=1,N
 5     A(ID)=0
       IF(L.EQ.1) GO TO 7
       L1=L-1
       DO 6 ID=1,L1
 6     LL=LL+I(ID)
 7     JJ=LL+1
       NB=LL+I(L)
       DO 8 ID=JJ,NB
 8     A(ID)=1
C---   PRINTING THE BLOCK
 9     PRINT 100, (A(II),II=1,N)
       LJ=LJ+1
       IF(LJ.GT.J(L)) GO TO 60
       GO TO 9
 60    L=L+1
       LJ=1
       IF(L.GT.M) GO TO 70
       GO TO 1
 70    LL=0
C---   PRINTING REMAINING &ZERO& ROWS, IF NECESSARY
       DO 10 ID=1,M
 10    LL=LL+I(ID)
       IF(LL.EQ.N) STOP
 12    LL=LL+1
       IF(LL.GT.N) STOP
       DO 11 ID=1,N
 11    A(ID)=0
       PRINT 100, (A(II),II=1,N)
       GO TO 12
 101   FORMAT (2I2,20I2)
 102   FORMAT (/1X,*THE SIZE OF THE MATRIX =*,I2/1X,*THE NO. OF
      1 BLOCKS IS=*,I2)
 103   FORMAT (/1X,* THE GENERATED MATRIX IS...*)
 100   FORMAT (/1X,30I2)
       END
£ENTRY
10040102020303010303
```

THE COMPUTER OUTPUT FOR THE TEST-DATA IS...

THE SIZE OF THE MATRIX = 10

THE NUMBER OF BLOCKS IN THE MATRIX = 4

THE GENERATED MATRIX IS...

```
1 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0
0 1 1 0 0 0 0 0 0 0
0 1 1 0 0 0 0 0 0 0
0 1 1 0 0 0 0 0 0 0
0 0 0 1 1 1 0 0 0 0
0 0 0 0 0 0 1 1 1 0
0 0 0 0 0 0 1 1 1 0
0 0 0 0 0 0 1 1 1 0
0 0 0 0 0 0 0 0 0 0
```

```
C----
C----  GENERATION OF BAND TRIANGULAR MATRIX
C----
C----  NOTATIONS ARE ...
C----  N=SIZE OF THE MATRIX
C----  M=NO. OF TRIANGLES
C----  J=ARRAY GIVING SIZES OF TRIANGLES
C----
       INTEGER A(30),B(30),J(10)
       READ 101, N,M,(J(K),K=1,M)
       PRINT 102,N,M
       PRINT 103
       L=1
       LJ=1
       K=1
       JJ=1
  1    LL=0
C----  CREATION OF THE FIRST ROW OF A TRIANGLE
       A(JJ)=1
       K2=JJ+1
       DO 7 ID=K2,N
  7    A(ID)=0
  8    PRINT 100, (A(II),II=1,N)
       LJ=LJ+1
       IF(LJ.GT.J(L)) GO TO 60
C----  SHIFT RIGHT OPERATION IS DONE BELOW
       DO 9 ID=1,N
  9    B(ID)=A(ID)
       DO 10 ID=2,N
       ID1=ID-1
  10   A(ID)=B(ID1)
       A(1)=0
       A(JJ)=1
       GO TO 8
  60   L=L+1
       IF(L.GT.M) GO TO 13
       LJ=1
       K1=K
       K=K+1
       DO 11 ID=1,K1
  11   LL=LL+J(ID)
       DO 12 ID=1,LL
  12   A(ID)=0
       JJ=LL+1
       GO TO 1
  13   LL=0
C----  ADD &ZERO& ROWS IF REQUIRED
       DO 14 ID=1,M
  14   LL=LL+J(ID)
       IF(LL.EQ.N) STOP
  16   LL=LL+1
       IF(LL.GT.N) STOP
       DO 15 ID=1,N
  15   A(ID)=0
```

```
      PRINT 100, (A(II),II=1, )
      GO TO 16
 101  FORMAT (1I2, CI2)
 102  FORMAT (/1X,*THE SIZE OF THE MATRIX =*,I2/1X,*NO. OF
     1 TRIANGLES =*,I2)
 103  FORMAT (/1X,* THE GENERATED MATRIX IS...*)
 100  FORMAT (/1X,3CI2)
      END
£ENTRY
100302020
```

THE COMPUTER OUTPUT FOR THE TEST=DATA IS...

```
THE SIZE OF THE MATRIX IS =10

NO. /F TRIANGLES = [illegible]

THE GENERATED MATRIX IS...

  1 0 0 0 0 0 0 0 0 0
  1 1 0 0 0 0 0 0 0 0
  0 0 1 0 0 0 0 0 0 0
  0 0 1 1 0 0 0 0 0 0
  0 0 0 0 1 0 0 0 0 0
  0 0 0 0 1 1 0 0 0 0
  0 0 0 0 1 1 1 0 0 0
  0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0
```

```
C---
C---   GENERATION OF &OVERLAPPING-BLOCKS& MATRIX
C---
C---   NOTATIONS ARE...
C---   N=SIZE OF THE MATRIX
C---   M=NO. OF BLOCKS
C---   I,J=ARRAYS GIVING SIZES OF BLOCKS
C---   K=ARRAY GIVING FIRST COLUMN OF BLOCKS
C---
       INTEGER A(30),I(10),J(10),K(10)
       READ 102, N,M
       READ 101, (I(L),J(L),K(L),L=1,M)
       PRINT 103,N,M
       PRINT 104
       L=1
       LJ=1
     1 CONTINUE
       DO 5 ID=1,N
  5    A(ID)=0
       NB=K(L)+I(L)-1
       KA=K(L)
       DO 6 ID=KA,NB
  6    A(ID)=1
  7    PRINT 100, (A(II),II=1,N)
       LJ=LJ+1
       IF(LJ.GT.J(L)) GO TO 60
       GO TO 7
 60    L=L+1
       LJ=1
       IF(L.GT.M) GO TO 70
       GO TO 1
C---   &ZERO ROWS GENERATED IF NECESSARY
 70    LL=0
       DO 10 ID=1,M
 10    LL=LL+J(ID)
       IF(LL.EQ.N) STOP
 12    LL=LL+1
       IF(LL.GT.N) STOP
       DO 11 ID=1,N
 11    A(ID)=0
       PRINT 100, (A(II),II=1,N)
       GO TO 12
 101   FORMAT (30I2)
 102   FORMAT (2I2)
 103   FORMAT (/1X,*THE SIZE OF THE MATRIX =*,I2//1X,*NO. OF
     1 BLOCKS =*,I2)
 104   FORMAT (/1X,*THE GENERATED MATRIX IS...*)
 100   FORMAT (/1X,30I2)
       END
£ENTRY
1003
0303010402030502O4
```

THE COMPUTER OUTPUT FOR THE TEST-DATA IS...

THE SIZE OF THE MATRIX IS = 10

NO. OF BLOCKS = 3

THE GENERATED MATRIX IS...

```
1 1 1 0 0 0 0 0 0 0
1 1 1 0 0 0 0 0 0 0
1 1 1 0 0 0 0 0 0 0
0 0 1 1 1 1 0 0 0 0
0 0 1 1 1 1 0 0 0 0
0 0 0 1 1 1 1 1 0 0
0 0 0 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

```
C---
C---  GENERATION OF MULTIPLE AND &OVERLAPPING BLOCKS& MATRIX
C---
C---  NOTATIONS ARE...
C---  N=SIZE OF THE MATRIX
C---  NDV=NO. OF SETS OF BLOCKS
C---  S,F=ARRAYS GIVING STARTING AND FINISHING COLS. OF
C---  DIFFERENT BLOCKS.
C---  NJ=NO. OF BLOCKS IN A SET
C---
      INTEGER A(30),M(15),S(15,5),F(15,5),NB(5),NJ(10)
      READ 101, N,NDV
      PRINT 105,N,NDV
      PRINT 106
      DO 1 I=1,NDV
      READ 102, M(I)
      KA=M(I)
      DO 2 J=1,KA
 2    READ 103, S(I,J),F(I,J)
 1    CONTINUE
      READ 104, (NJ(II),II=1,NDV)
      L=1
      LJ=1
 15   CONTINUE
      DO 5 ID=1,N
 5    A(ID)=0
      KA=M(L)
      DO 6 K=1,KA
      NB(K)=S(L,K)+F(L,K)-1
      IA=S(L,K)
      IB=NB(K)
      DO 7 ID=IA,IB
 7    A(ID)=1
 6    CONTINUE
 8    PRINT 100, (A(II),II=1,N)
      LJ=LJ+1
      IF(LJ.GT.NJ(L)) GO TO 60
      GO TO 8
 60   L=L+1
      LJ=1
      IF(L.GT.NDV) GO TO 70
      GO TO 15
 70   LL=0
C---  GENERATE &ZERO& ROWS IF NECESSARY
      DO 10 ID=1,NDV
 10   LL=LL+NJ(ID)
      IF(LL.EQ.N) STOP
 12   LL=LL+1
      IF(LL.GT.N) STOP
      DO 11 ID=1,N
 11   A(ID)=0
      PRINT 100, (A(II),II=1,N)
      GO TO 12
 101  FORMAT (2I2)
```

```
 102   FORMAT (I2)
 103   FORMAT (2I2)
 104   FORMAT (10I2)
 105   FORMAT (/1X,*THE SIZE OF THE MATRIX =*,I2/1X,*NO. OF
     1  SETS OF BLOCKS =*,I2)
 106   FORMAT (/1X,* THE GENERATED MATRIX IS...*)
 100   FORMAT (/1X,30I2)
       END
£ENTRY
1003
02
0103
0609
03
0103
0405
0709
01
0207
030204
```

THE COMPUTER OUTPUT FOR THE TEST-DATA IS...

```
THE SIZE OF THE MATRIX =10
NO. OF SETS OF BLOCKS= 3

THE GENERATED MATRIX IS...

  1 1 1 0 0 1 1 1 1 1
  1 1 1 0 0 1 1 1 1 1
  1 1 1 0 0 1 1 1 1 1
  1 1 1 1 1 1 1 1 1 1
  1 1 1 1 1 1 1 1 1 1
  0 1 1 1 1 1 1 1 0 0
  0 1 1 1 1 1 1 1 0 0
  0 1 1 1 1 1 1 1 0 0
  0 1 1 1 1 1 1 1 0 0
  0 0 0 0 0 0 0 0 0 0
```

```
C---
C---   GENERATION OF V-SHAPED MATRIX
C---
C---   NOTATIONS ARE...
C---   N=SIZE OF THE MATRIX
C---   I1=NO. OF COLUMNS IN A SIDE OF V
C---   J1=NO. OF ROWS FORMING V-BLOCK
C---   A,B=ARRAYS, WHICH WHEN CONCATENATED, GIVE A ROW OF
C---   DESIRED MATRIX.
C---
       INTEGER A(15),B(15),C(15),D(15)
       READ 101, N,I1,J1
       PRINT 102,N,I1,J1
       PRINT 103
       KK=0
       N1=N/2
       IF(N-N1*2.EQ.1) GO TO 1
       K=N/2
       L=K
       GO TO 2
1      K=N/2+1
       L=K-1
2      CONTINUE
       M=1
       DO 3 I=1,K
       A(I)=0
       IF(I.LE.I1) A(I)=1
3      CONTINUE
       DO 4 J=1,L
       B(J)=0
       IF(J.GE.L-I1+1) B(J)=1
4      CONTINUE
9      PRINT 100, (A(I),I=1,K),(B(J),J=1,L)
       IF(M.EQ.N) STOP
       M=M+1
       IF(M.GT.J1) KK=1
       DO 5 I=1,K
5      C(I)=A(I)
       DO 6 J=1,L
6      D(J)=B(J)
       DO 7 I=2,K
       II=I-1
7      A(I)=C(II)
       L1=L-1
       DO 8 J=1,L1
       JJ=J+1
8      B(J)=D(JJ)
       IF(KK.EQ.0) GO TO 10
       A(1)=0
       B(L)=0
       GO TO 9
10     A(1)=1
       B(L)=1
       GO TO 9
```

```
 101   FORMAT ( I2)
 102   FORMAT (/1X,*THE SIZE OF THE MATRIX =*,I2/1X,*NO. OF
     1 COLS. IN HALF-V =*,I2/1X,*NO. OF ROWS FORMING V-BLOCK
     1 =*,I2)
 103   FORMAT (/1X,* THE GENERATED MATRIX IS...*)
 100   FORMAT (/1X,10I2)
       END
£ENTRY
100307
```

THE COMPUTER OUTPUT FOR THE TEST-DATA IS...

```
THE SIZE OF THE MATRIX =10
NO. OF COLUMNS IN HALF-V = 3
NO. OF ROWS FORMING V-BLOCK = 7

THE GENERATED MATRIX IS...

  1 1 1 0 0 0 0 1 1
  1 1 1 1 0 0 1 1 1
  1 1 1 1 1 1 1 1 1
  1 1 1 1 1 1 1 1 1
  1 1 1 1 1 1 1 1 1
  1 1 1 1 1 1 1 1 1
  1 1 1 1 1 1 1 1 1
  0 1 1 1 1 1 1 1 1 0
  0 0 1 1 1 1 1 1 0 0
  0 0 0 1 1 1 1 0 0 0
```

```
C---
C-----CONVERSION OF A GIVEN MATRIX INTO BAND MATRIX FORM.
C--
      INTEGER A(15,15),R(15),C(15),SR(15),SC(15),TAGR(15),
     1 B(15,15),MSC(15)
C******NOTATIONS...
C-----N=SIZE OF THE MATRIX
C-----A=INPUT MATRIX
C-----R=ROW SUM VECTOR
C-----C=COLUMN SUM VECTOR
C-----MAXR=MAXIMUM OF THE ROW SUMS
C-----MAXC=MAXIMUM OF THE COLUMN SUMS
C-----MINR=MINIMUM OF THE ROW SUMS
C-----MINC=MINIMUM OF THE COLUMN SUMS
C-----SR=SORTED ROW VECTOR
C-----SC=SORTED COLUMN VECTOR
C*****
      READ 101,N
      PRINT 102
      DO 1 I=1,N
      READ 103,(A(I,J),J=1,N)
      PRINT 104,(A(I,J),J=1,N)
 1    CONTINUE
C-----COMPUTATION OF ROW SUMS AND COLUMN SUMS
      DO 5 I=1,N
      R(I)=0
      TAGR(I)=0
      DO 5 J=1,N
 5    R(I)=R(I)+A(I,J)
      DO 10 J=1,N
      C(J)=0
      DO 10 I=1,N
 10   C(J)=C(J)+A(I,J)
C-----DETERMINATION OF MAXIMUN AND MINIMUM OF ROW AND COLUMN SUMS
      MAXR=0
      MINR=20
      MAXC=0
      MINC=20
      DO 15 I=1,N
      IF(MAXR.LT.R(I)) MAXR=R(I)
      IF(MINR.GT.R(I)) MINR=R(I)
      IF(MAXC.LT.C(I)) MAXC=C(I)
      IF(MINC.GT.C(I)) MINC=C(I)
 15   CONTINUE
C---  CHECK TO SEE WHETHER CONDITION OF BAND DIAGONALITY
C---  IS SATISFIED.
      ICS=0
      CALL CONCHK(R,C,MAXR,MAXC,N,ICS)
      IF(ICS.EQ.1) GO TO 500
C-----ARRANGE ROWS AND COLUMNS (ROWS FIRST) AS THEY WOULD
C     APPEAR IN THE BAND MATRIX.
 45   L=1
 46   INDEX=0
      CALL MNRWSM(R,MINR,INDEX,SC,N,TAGR,IR)
```

```
      ICS=0
      CALL ROWFIN(MINR,MAXR,R,A,IR,INDEX,SC,TAGR,N,ICS)
      IF(ICS.EQ.1) GO TO 500
      CALL RWMAXR(N,MAXR,A,R,TAGR,IR,INDEX,SC)
      IF(ICS.EQ.1) GO TO 500
      IF(ICS.EQ.2) GO TO 100
      CALLRWDCRS(A,R,TAGR,SC,IR,MAXR,N,INDEX)
      IF(ICS.EQ.1) GO TO 500
100   L=L+1
      IF(L.GT.2) GO TO 300
      IRD=MAXR-MINR
      MAXR=MAXC
      MINR=MINC
      DO 105 I=1,N
      R(I)=C(I)
      TAGR(I)=0
      SR(I)=SC(I)
      DO 105 J=1,N
105   B(I,J)=A(I,J)
      DO 110 I=1,N
      DO 110 J=1,N
110   A(I,J)=B(J,I)
      GO TO 46
300   CONTINUE
C---- REVERSE COLUMN ORDER IF NECESSARY
      IF(MINC.EQ.IRD+1) GO TO 310
      DO 115 I=1,N
115   MSC(I)=SC(I)
      DO 120 I=1,N
      J=N-I+1
120   SC(I)=MSC(J)
310   CONTINUE
C-----PREPARATION FOR FINAL PRINTING
      DO 125 I=1,N
      DO 125 J=1,N
      IR=SR(I)
      IC=SC(J)
125   A(I,J)=B(IR,IC)
      PRINT 106
      DO 140 I=1,N
140   PRINT 104,(A(I,J),J=1,N)
      STOP
500   PRINT 108
      STOP
101   FORMAT (I2)
102   FORMAT (1X,*THE GIVEN MATRIX*//)
103   FORMAT (15I1)
104   FORMAT (1X,15I2)
106   FORMAT (//,1X,*THE MATRIX IN REQUIRED FORM*,//)
08    FORMAT (1X,*THE GIVEN MATRIX CANNOT BE CONVERTED
     1 TO BAND DIAGONAL FORM*)
      END
      SUBROUTINE CONCHK(R,C,MAXR,MAXC,N,ICS)
C----
```

```
C---  THIS ROUTINE CHECKS WHETHER CONDITION OF BAND
C---  DIAGONALITY IS SATISFIED OR NOT. IF YES, THEN ICS=0.
C---  53SE, ICS=1.
C---
      INTEGER R(15),C(15)
      K2=MAXR
      K3=MAXC
      IFLR=0
      IFLC=0
      IFLR1=0
      IFLC1=0
      KOUNT1=0
      KOUNT2=0
      DO 20 I=1,N
      IF(R(I).EQ.MAXR) KOUNT1=KOUNT1+1
      IF(C(I).EQ.MAXC) KOUNT2=KOUNT2+1
 20   CONTINUE
      IF(KOUNT1.NE.N-MAXR+1.OR.KOUNT2.NE.N-MAXC+1) GO TO 500
 25   IF(N-KOUNT1.EQ.0) GO TO 35
      IF(N-KOUNT1.EQ.1) IFLR=
      KOUNT=0
      K2=K2-1
      DO 30 I=1,N
      IF(R(I).NE.K2) GO TO 30
      KOUNT=KOUNT+1
      KOUNT1=KOUNT1 +1
 30   CONTINUE
      IF(IFLR1.EQ.1.AND.KOUNT.NE.1) GO TO 500
      IF(IFLR.EQ.0.AND.KOUNT.EQ.2) GO TO 25
      IF(IFLR.EQ.1.AND.KOUNT.EQ.1) GO TO 35
      IF(IFLR.EQ.0.AND.KOUNT.EQ.1) GO TO 31
      GO TO 500
 31   IFLR1=1
      GO TO 25
 35   IF(N-KOUNT2.EQ.0) GO TO 45
      IF(N-KOUNT2.EQ.1) IFLC=1
      KOUNT=0
      K3=K3-1
      DO 40 I=1,N
      IF(C(I).NE.K3) GO TO 40
      KOUNT=KOUNT+1
      KOUNT2=KOUNT2+1
 40   CONTINUE
      IF(IFLC1.EQ.1.AND.KOUNT.NE.1) GO TO 500
      IF(IFLC.EQ.0.AND.KOUNT.EQ.2) GO TO 35
      IF(IFLC.EQ.1.AND.KOUNT.EQ.1) GO TO 45
      IF(IFLC.EQ.0.AND.KOUNT.EQ.1) GO TO 41
      GO TO 500
 41   IFLC1=1
      GO TO 35
 500  ICS=1
 45   RETURN
      END
      SUBROUTINE MNRWSM(R,MINR,INDEX,SC,N,TAGR,IR)
```

```
C---
C---   THIS ROUTINE CHOSES ROW WHOSE ROW-SUM IS MINIMUM.
C---
      INTEGER R(15),SC(15),TAGR(15)
      DO 50 I=1,N
      IF(R(I).NE.MINR) GO TO 50
      IR=I
      INDEX=INDEX+1
      SC(INDEX)=I
      TAGR(I)=1
      RETURN
 50   CONTINUE
      END
      SUBROUTINE ROWFIN(MINR,MAXR,R,A,IR,INDEX,SC,TAGR,N,ICS)
C---
C---   THIS ROUTINE FINDS FIRST FEW ROWS OF MATRIX WHOSE
C---   ROW-SUMS INCREASE FROM MINR TO MAXR IN STEPS OF 1. IF SUCH
C---   ROWS ARE NOT FOUND, ICS=1, ELSE ICS=0
C---
      INTEGER A(15,15),R(15),SC(15),TAGR(15)
 55   IP=MINR
 56   CONTINUE
      IF(IP.EQ.MAXR) GO TO 70
      DO 60 I=1,N
      IF(R(I).NE.IP+1) GO TO 60
      KOUNT=0
      DO 65 J=1,N
      IF((A(IR,J).EQ.0.AND.A(I,J).EQ.1).OR.(A(IR,J).EQ.1.AND.
     1 A(I,J).EQ.0)) KOUNT=KOUNT+1
 65   CONTINUE
      IF(KOUNT.NE.1) GO TO 60
      IP=IP+1
      IR=I
      INDEX=INDEX+1
      SC(INDEX)=I
      TAGR(I)=1
      GO TO 56
 60   CONTINUE
      ICS=1
 70   RETURN
      END
      SUBROUTINE RWMAXR(N,MAXR,A,R,TAGR,IR,INDEX,SC)
C---
C---   THIS ROUTINE FINDS (N-MAXR) ROWS WHOSE ROW-SUM=MAXR, AND
C---   DISTANCE FROM PREVIOUS ROW=2. IF FOUND, ICS=0. IF NOT, ICS=1.
C---   IF ALL ROWS OVER, ICS=2.
C---
      INTEGER A(15,15),R(15),TAGR(15),SC(15)
 70   K=1
 71   IF(K.GT.N-MAXR) GO TO 85
      DO 75 I=1,N
      IF(R(I).NE.MAXR) GO TO 75
      IF(TAGR(I).NE.0) GO TO 75
      KOUNT=0
```

```
      DO 80 J=1,N
      IF((A(IR,J).EQ.0.AND.A(I,J).EQ.1).OR.(A(IR,J).EQ.1.AND.
     1 A(I,J).EQ.0)) KOUNT=KOUNT+1
 80   CONTINUE
      IF(KOUNT.NE.2) GO TO 75
      K=K+1
      IR=I
      INDEX=INDEX+1
      SC(INDEX)=I
      TAGR(I)=1
      IF(INDEX.EQ.N) GO TO 100
      GO TO 71
 75   CONTINUE
      ICS=1
      GO TO 85
100   ICS=2
 85   RETURN
      END
      SUBROUTINE RWDCRS(A,R,TAGR,SC,IR,MAXR,N,INDEX)
C---
C---  THIS ROUTINE FINDS LOWER PORTION OF THE MATRIX WITH ROWS
C---  HAVING DECREASING ROW-SUMS IN STEPS OF 1 AND DISTANCE BETWEEN
C---  CONSECUTIVE ROWS =1. IF SUCH ROWS NOT FOUND, ICS=1, ELSE ICS=0
C---
      INTEGER A(15,15),R(15),TAGR(15),SC(15)
 85   IP=MAXR-1
 86   CONTINUE
      DO 90 I=1,N
      IF(R(I).NE.IP.OR.TAGR(I).NE.0) GO TO 90
      KOUNT=0
      DO 95 J=1,N
      IF((A(IR,J).EQ.0.AND.A(I,J).EQ.1).OR.(A(IR,J).EQ.1.AND.
     1 A(I,J).EQ.0)) KOUNT=KOUNT+1
 95   CONTINUE
      IF(KOUNT.NE.1) GO TO 90
      IP=IP-1
      IR=I
      INDEX=INDEX+1
      SC(INDEX)=I
      TAGR(I)=1
      IF(INDEX.EQ.N) GO TO 100
      GO TO 86
 90   CONTINUE
      ICS=1
100   RETURN
      END
£ENTRY
07
1001001
0010011
1100100
0010010
1001100
0011001
```

0000010

THE COMPUTER OUTPUT FOR THE TEST-DATA IS...

THE GIVEN MATRIX

```
1 0 0 1 0 0 1
0 0 1 0 0 1 1
1 1 0 0 1 0 0
0 0 1 0 0 1 0
1 0 0 1 1 0 0
0 0 1 1 0 0 1
0 0 0 0 0 1 0
```

THE MATRIX IN REQUIRED FORM

```
1 0 0 0 0 0 0
1 1 0 0 0 0 0
1 1 1 0 0 0 0
0 1 1 1 0 0 0
0 0 1 1 1 0 0
0 0 0 1 1 1 0
0 0 0 0 1 1 1
```

```
C---
C---  CONVERSION OF A GIVEN MATRIX INTO A BLOCK MATRIX
C---  ASSUMED THAT A COLUMN IS NOT COMMON TO MORE THAN TWO BLOCKS.
C---
      INTEGER A(15,15),B(15,15),R(15),C(15),TAGR(15),TAGC(15),
     1 BLR(10,5),BLC(10,5),NM(10),FLAG(10),SB(10),SR(15),SC(15)
      INTEGER KT(10),F(10,10)
C***  NOTATIONS ARE AS FOLLOWS
C---  A=INPUT MATRIX
C---  B=DUMMY MATRIX
C---  R=ROW SUM VECTOR
C---  C=COLUMN SUM VECTOR
C---  TAGR=ROW TAGS
C---  TAGC=COLUMN TAGS
C---  BLR=ROW FORMING A BLOCK
C---  BLC=COLUMNS FORMING A BLOCK
C---  SB=SORTED BLOCKS
C---  SR=SORTED ROWS
C---  SC=SORTED COLUMNS
C---
      READ 101,N
      PRINT 102
      L=0
      M=0
      ICH=0
      M1=0
      DO 1 I=1,N
      R(I)=0
      C(I)=0
      TAGR(I)=0
      TAGC(I)=0
      READ 103,(A(I,J),J=1,N)
    1 PRINT 104,(A(I,J),J=1,N)
C---  CALCULATE ROW AND COLUMN SUMS
      DO 5 I=1,N
      DO 6 J=1,N
      R(I)=R(I)+A(I,J)
      C(J)=C(J)+A(I,J)
    6 B(I,J)=A(I,J)
    5 IF(R(I).EQ.0) TAGR(I)=1
      DO 7 I=1,10
      DO 7 J=1,5
      BLC(I,J)=0
    7 BLR(I,J)=0
      DO 8 I=1,10
      DO 8 J=1,10
    8 F(I,J)=0
      CALL BLKFOR(A,TAGR,N,L,M,M1,BLR,BLC,R,ICH)
C---  NUMBER OF BLOCKS FORMED=L
      INDEX=0
      CALL BLKINT(L,F,BLC)
C---  CHECK FOR BLOCK INTERSECTION AND IF IT IS GREATER THAN 2,
C---  REPORT FAILURE. OTHERWISE CHOOSE FIRST BLOCK.
C---
```

```
      DO 53 I=1,L
      FLAG(I)=0
      KT(I)=0
      DO 54 J=1,L
      IF(I.EQ.J) GO TO 54
      IF(F(I,J).EQ.0) GO TO 54
      KT(I)=KT(I)+1
 54   CONTINUE
 53   CONTINUE
 55   IB=0
      DO 57 I=1,L
      IF(FLAG(I).NE.0) GO TO 57
      IF(KT(I).GT.2) GO TO 500
      IF(KT(I).GT.1) GO TO 57
      IB=I
 57   CONTINUE
      IF(IB.EQ.0) GO TO 500
C---  IB POINTS TO FIRST BLOCK
      INDEX=INDEX+1
      SB(INDEX)=IB
      FLAG(IB)=1
      IF(INDEX.EQ.L) GO TO 100
      CALL BLKORD(IC,F,L,KT,INDEX,SB,FLAG,IB)
      IF(IC.EQ.1) GO TO 55
C---  SB CONTAINS BLOCKS IN ORDER OF PRINTING
 100  INDEXR=0
      INDEXC=0
      IP=1
      CALL BLKARG(INDEXR,INDEXC,IP,SB,L,BLC,BLR,TAGC,SC,SR)
 155  CONTINUE
C---  PREPARATION FOR FINAL PRINTING. INCLUDE &ZERO& ROWS
C---  AND COLS. , IF REQUIRED.
C---
      DO 160 I=1,N
      IF(I.GT.INDEXR) GO TO 165
      IR=SR(I)
      DO 170 J=1,N
      IF(J.GT.INDEXC) GO TO 160
      IC=SC(J)
 170  A(I,J)=B(IR,IC)
 160  CONTINUE
 165  CONTINUE
      IF(N.EQ.INDEXR) GO TO 175
      KR=N-INDEXR
      DO 180 I=1,KR
      IX=I+INDEXR
      DO 185 J=1,N
 185  A(IX,J)=0
 180  CONTINUE
 175  CONTINUE
      IF(N.EQ.INDEXC) GO TO 190
      KC=N-INDEXC
      DO 195 I=1,KC
      IY=I+INDEXC
```

```
      DO 200 J=1,N
200   A(J,IY)=0
195   CONTINUE
190   CONTINUE
      PRINT 106
      DO 205 I=1,N
205   PRINT 104,(A(I,J),J=1,N)
      STOP
500   PRINT 108
      STOP
101   FORMAT (I2)
102   FORMAT (1X,*THE INPUT MATRIX*//)
103   FORMAT (15I1)
104   FORMAT (1X,15I2)
106   FORMAT (//,1X,* THE MATRIX IN REQUIRED FORM*,//)
108   FORMAT (1X,*THE GIVEN MATRIX CANNOT BE CONVERTED TO
     1A BLOCK MATRIX*)
      END
      SUBROUTINE BLKFOR(A,TAGR,N,L,M,M1,BLR,BLC,R,ICH)
C===
C===  THIS ROUTINE FORMS BLOCKS AND STORES ROWS AND COLS.
C===  FORMING BLOCK I, IN BLR(I,..) AND BLC(I,..) RESPECTIVELY.
C===
      INTEGER A(15,15),TAGR(15),BLR(10,5),BLC(10,5),R(15)
      N1=N-1
      DO 10 I=1,N1
      IF(TAGR(I).NE.0) GO TO 10
      L=L+1
      M1=M1+1
      BLR(L,M1)=I
      TAGR(I)=1
      I1=I+1
      DO 15 J = I1,N
      IF(TAGR(J).NE.0) GO TO 15
      IF(R(I).NE.R(J)) GO TO 15
      DO 20 K=1,N
      IF(A(I,K)-A(J,K).NE.0) GO TO 15
20    CONTINUE
      TAGR(J)=1
      M1=M1+1
      BLR(L,M1)=J
      DO 25 K=1,N
      IF(A(I,K).NE.1) GO TO 25
      M=M+1
      BLC(L,M)=K
      ICH=1
25    CONTINUE
      M=0
      M1=0
15    CONTINUE
      IF(ICH.EQ.1) GO TO 35
      DO 30 K=1,N
      IF(A(I,K).NE.1) GO TO 30
      M=M+1
```

```
      BLC(L,M)=K
 30   CONTINUE
      M=0
      M1=0
      GO TO 10
 35   ICH=0
 10   CONTINUE
      RETURN
      END
      SUBROUTINE BLKINT(L,F,BLC)
C---
C---  THIS ROUTINE FINDS BLOCKS HAVING AT LEAST A COL. IN COMMON
C---  BETWEEN THEM. IF BLOCKS I AND J INTERSECT, F(I,J)=F(J,I)=1
C---
      INTEGER F(10,10),BLC(10,5)
      L1=L-1
      DO 40 I=1,L1
      I1=I+1
      DO 45 J=I1,L
      DO 50 K=1,5
      IF(BLC(I,K).EQ.0) GO TO 45
      DO 52 M=1,5
      IF(BLC(J,M).EQ.0) GO TO 50
      IF(BLC(I,K).NE.BLC(J,M)) GO TO 52
      F(I,J)=1
      F(J,I)=1
      GO TO 45
 52   CONTINUE
 50   CONTINUE
 45   CONTINUE
 40   CONTINUE
      RETURN
      END
      SUBROUTINE BLKORD(IC,F,L,KT,INDEX,SB,FLAG,IB)
C---
C---  THIS ROUTINE ARRANGES BLOCKS IN ORDER IN WHICH THEY ARE
C---  GOING TO BE PRINTED. IC=1 IMPLIES A NEW BLOCK IS TO BE FOUND
C---  AS PREVIOUS BLOCKS DO NOT INTERSECT WITH REMAINING BLOCKS.
C---  IC=2 IMPLIES BLOCKS ARE ARRANGED PROPERLY.
C---
      INTEGER F(10,10),KT(10),SB(10),FLAG(10)
 59   IC=0
      DO 60 J=1,L
      IF(F(IB,J).EQ.0) GO TO 60
      F(IB,J)=0
      F(J,IB)=0
      KT(IB)=KT(IB)-1
      KT(J)=KT(J)-1
      IC=J
      INDEX=INDEX+1
      SB(INDEX)=J
      FLAG(J)=1
      IB=J
      IF(INDEX.EQ.L) GO TO 100
```

```
      GO TO 59
 60   CONTINUE
      IC=1
      RETURN
 100  IC=2
      RETURN
      END
      SUBROUTINE BLKARG(INDEXR,INDEXC,IP,SB,L,BLC,BLR,TAGC,SC,SR)
C---
C---  THIS ROUTINE ARRANGES ROWS AND COLS. FROM SB-VECTOR IN
C---  PROPER ORDER FOR FINAL PRINTING.
C-
      INTEGER SB(10),BLC(10,5),BLR(10,5),TAGC(15),SC(15),SR(15)
 99   KA=SB(IP)
      IP1=IP+1
      IF(IP1.EQ.L+1) GO TO 115
      KB=SB(IP1)
      DO 110 I=1,5
      IF(BLC(KA,I).EQ.0) GO TO 115
      DO 120 J=1,5
      IF(BLC(KA,I).NE.BLC(KB,J)) GO TO 120
      IZ=BLC(BA,I)
      TAGC(IZ)=1
      GO TO 110
 120  CONTINUE
 110  CONTINUE
 115  CONTINUE
      DO 125 I=1,5
      IF(BLC(KA,I).EH=0)+GO TO 130
      IZ=BLC(KA,I)
      IF(TAGC(IZ).NE.0) GO TO 125
      INDEXC=INDEXC+1
      SC(INDEXC)=IZ
 125  CONTINUE
 130  CONTINUE
      DO 135 I=1,5
      IF(BLC(KA,I).EQ.0) GO TO 140
      IZ=BLC(KA,I)
      IF(TAGC(IZ).NE.1) GO TO 135
      INDEXC=INDEXC+1
      SC(INDEXC)=IZ
 135  CONTINUE
 140  CONTINUE
      DO 145 I=1,5
      IF(BLR(KA,I).EQ.0) GO TO 150
      INDEXR=INDEXR+1
 145  SR(INDEXR)=BLR(KA,I)
 150  IP=IP+1
      IF(IP.GT.L) GO TO 155
      DO 151 I=1,INDEXC
      II=SC(I)
 151  TAGC(II)=2
      GO TO 99
 155  RETURN
```

```
     END
£ENTRY
07
0000110
1010001
0000000
0000001
1010001
0000001
0000110
```

THE COMPUTER OUTPUT FOR THE TEST-DATA IS...

THE INPUT MATRIX

```
0 0 0 0 1 1 0
1 0 1 0 0 0 1
0 0 0 0 0 0 0
0 0 0 0 0 0 1
1 0 1 0 0 0 1
0 0 0 0 0 0 1
0 0 0 0 1 1 0
```

THE MATRIX IN REQUIRED FORM

```
1 0 0 0 0 0 0
1 0 0 0 0 0 0
1 1 1 0 0 0 0
1 1 1 0 0 0 0
0 0 0 1 1 0 0
0 0 0 1 1 0 0
0 0 0 0 0 0 0
```

```
C---
C-----CONVERSION OF A GIVEN MATRIX INTO BAND TRIANGULAR FORM.
C---
C***   NOTATIONS ARE AS FOLLOWS
C---   A=INPUT MATRIX
C---   R=ROW SUM VECTOR
C---   TAGR=ROW TAGS
C---   TAGC=COLUMN TAGS
C---   SR=SORTED ROW VECTOR
C---   P=DUMMY VECTOR
C---   B=DUMMY MATRIX
C---
       INTEGER A(15,15),R(15),TAGR(15),TAGC(15),SR(15),P(15),
      1 B(15,15)
C---INITIALIZATION
C---
       95A4 101,N
       7RINT 102
       DO 1 I=1,N
       98I)=0
       TAGR(I)=0
       TIGC(I)=0
       R5AD 103,(A(I,J),J=1,N)
 1     7RINT 104,(A(I,J),J=1,N)
        9=0
C--    3AL3UL1TE ROW SUMS
       46 2 I=1.N
       40 5 J=1,N
 5     R(I)=R(I)+A(I,J)
 2     96(R(I&.EQ.0) KR=KR+1
       IF(KR.EQ.0) GO TO 3
 4     36NTIN4E
C---   PUT &ZERO& ROWS AT THE END OF SR VECTOR
       KK=0
       DO 7 I=1,N
       IF(R(I).NE.0) GO TO 7
       KK=KK+1
       II=N-KK+1
       SR(II)=I
 7     CONTINUE
 3     INDEX=0
 6     CONTINUE
       DO 10 I=1,N
C---   CHOOSE A ROW WHOSE ROW-SUM=1 AND WHICH IS NOT YET
C--    CONSIDERED.
       IF(R(I).NE.1.OR.TAGR(I).NE.0) GO TO 10
       DO 15 J=1,N
       IF(A(I,J).NE.1) GO TO 15
       IF(TAGC(J).NE.0) GO TO 10
       TAGR(I)=1
       INDEX=INDEX+1
       TAGC(INDEX)=1
       SR(INDEX)=I
       99.9
```

```
C---    INTERCHANGE COLUMNS J AND INDEX
        DO 20 K=1,N
        P(K)=A(K,J)
        A(K,J)=A(K,INDEX)
 20     A(K,INDEX)=P(K)
        IF(INDEX.EQ.N-KR) GO TO 100
        GO TO 25
 15     CONTINUE
 10     CONTINUE
        GO TO 500
 25     K=2
 26     CONTINUE
C---    CHOOSE A ROW HAVING ROW SUM=K AND WHICH IS NOT
C--     8ET CONSIDERED.
        DO 30 I=1,N
        IF(R(I).NE.K.OR.TAGR(I).NE.0) GO TO 30
        KOUNT=0
C---    CALCULATE DISTANCE BETWEEN ROWS I AND IR
        DO 35 J=1,N
        IF((A(I,J).EQ.0.AND.A(IR,J).EQ.1).OR.(A(I,J).EQ.1.AND.
       1 A(IR,J).EQ.0)) KOUNT=KOUNT+1
 35     CONTINUE
        IF(KOUNT.NE.1) GO TO 30
        DO 40 J=1,N
        96(A(I,J).NE.1.OR.TAGC(J).NE.0) GO TO 40
        IF(A(IR,J).EQ.1) GO TO 40
        TAGR(I)=1
        INDEX=INDEX+1
        TAGC(INDEX)=1
        SR(INDEX)=I
        IR=I
        K=K+1
C---    INTERCHANGE COLUMNS L AND INDEX
        DO 45 L=1,N
        P(L)=A(L,J)
        A(L,J)=A(L,INDEX)
 45     A(L,INDEG&=P(L)
        IF(INDEX.EQ.N-KR) GO TO 100
        GO TO 26
 40     CONTINUE
 30     CONTINUE
        GO TO 6
 100    CONTINUE
        DO 50 I=1,N
        DO 50 J=1.N
 50     B(I,J)=A(I,J)
C---    PERMUTE ROWS ACCORDING TO SR-VECTOR
        DO 55 I=1,N
        IR=SR(I)
        DO 55 J=1,N
 55     A(I,J)=B(IR,J)
        PRINT 105
        DO 60 I=1,N
 60     PRINT 10?,(A(I,J),J=1,N)
```

```
      STOP
 500  PRINT 105
      STOP
 101  FORMAT (I2)
 102  FORMAT (1X,*THE GIVEN MATRIX*//)
 103  FORMAT (15I1)
 104  FORMAT (1X,15I2)
 105  FORMAT (1X,*THE GIVEN MATRIX CANNOT BE CONVERTED
     1 TO BAND TRIANGULAR FORM.*)
 106  FORMAT (1X,//,* THE MATRIX IN REQUIRED FORM*,//)
      END
£ENTRY
07
0100011
1010000
0000001
0100001
1000000
0000000
0001000
```

THE COMPUTER OUTPUT FOR THE TEST-DATA IS...

```
THE GIVEN MATRIX

  0 1 0 0 0 1 1
  1 0 1 0 0 0 0
  0 0 0 0 0 0 1
  0 1 0 0 0 0 1
  1 0 0 0 0 0 0
  0 0 0 0 0 0 0
  0 0 0 1 0 0 0


THE MATRIX IN REQUIRED FORM

  1 0 0 0 0 0 0
  1 1 0 0 0 0 0
  1 1 1 0 0 0 0
  0 0 0 1 0 0 0
  0 0 0 1 1 0 0
  0 0 0 0 0 1 0
  0 0 0 0 0 0 0
```

```
C===
C===   PROGRAM TO CONVERT A GIVEN RANDOM MATRIX TO BAND
C===   DIAGONAL FORM SUCH THAT A(I,J)=0 IF ABS(I-J) IS GREATER THAN
C===   BAND WIDTH OF THE CONVERTED MATRIX.
C===
C===   NOTATIONS USED ARE AS FOLLOWS...
C===   A= INPUT MATRIX
C===   N= SIZE OF THE MATRIX
C===   B= ROW COL. INCIDENCE MATRIX. UPPER TRIANGULAR PORTION
C===   FOR ROW AND LOWER ONE FOR COL. INCIDENCES.
C===   F AND DUM= DUMMY MATRICES
C===   R AND C= ROW AND COL. SUM VECTORS
C===   DR AND DC= ROW AND COL. DEGREE VECTORS
C===   SR AND SC= SORTED ROW AND COL. VECTORS
C===   TAGR AND TAGC= TAG VECTORS FOR ROWS AND COLS.
C===   ICL= FLAG INDICATING WHETHER ROWS OR COLS. ARE
C===   BEING PERMUTED.
C===IFF= FLAG TO INDICATE WHETHER ROW OR COL. IS BEING
C===   REMOVED FOR REDUCTION OF BAND-WIDTH.
C===   MAXM= MAXIMUM NO. OF NON-ZERO ENTRIES IN EITHER A ROW OR A COL.
C===   IRW= ROW OR COL. WHICH IS TO BE REMOVED.
      INTEGER A(15,15),B(15,15),F(15,15),DUM(15,15),R(15),C(15),
     1 DR(15),DC(15),SR(15),SC(15),TAGR(15),TAGC(15)
      INTEGER DUM1(15,15)
      READ 101,N
      PRINT 102
      DO 1 I=1,N
      READ 103,(A(I,J),J=1,N)
    1 PRINT 104,(A(I,J),J=1,N)
C===
C===   INITIALIZATION PHASE
C===
      IFF=0
      DO 2 I=1,N
      DO 2 J=1,N
    2 DUM(I,J)=A(I,J)
    3 ICL=0
      DO 5 I=1,N
      R(I)=0
      C(I)=0
      TAGR(I)=0
      TAGC(I)=0
      DR(I)=0
      DC(I)=0
      DO 5 J=1,N
      B(I,J)=0
    5 F(I,J)=A(I,J)
      DO 10 I=1,N
      DO 10 J=1,N
      R(I)=R(I)+A(I,J)
   10 C(J)=C(J)+A(I,J)
      IF(IFF.NE.0) GO TO 13
      MAXM=0
      DO 12 I=1,N
```

```
      IF(MAXM.LT.R(I)) MAXM=R(I)
      IF(MAXM.LT.C(I)) MAXM=C(I)
12    CONTINUE
13    CONTINUE
      CALL ROWINC(A,B,N,DR)
      CALL COLINC(A,B,N,DC)
      CALL NMBRWS(A,B,N,R,DR,SR,TAGR,ICL,SC)
      PRINT 107,(SR(I),I=1,N)
      N1=N-1
      DO 7 I=1,N1
      I1=I+1
      DO 7 J=I1,N
7     B(I,J)=B(J,I)
      CALL FRSCOL(SR,A,N,IR,TAGC)
      SC(1)=IR
      CALL NMBCOL(A,B,N,F,DC,SC,TAGC,ICL,C,SR)
      PRINT 105
      DO 14 I=1,N
      DO 14 J=1,N
14    DUM1(I,J)=DUM(I,J)
      DO 15 I=1,N
      IR1=SR(I)
      DO 15 J=1,N
      IC1=SC(A)
      DUM(I,J)=DUM1(IR1,IC1)
15    A(I,J)=F(IR1,IC1)
      DO 20 I=1,N
20    PRINT 104,(A(I,J),J=1,N)
      CALL RWCLRM(A,MAXM,N,IRW,IFF)
      IF(IRW.EQ.0) GO TO 25
      9FF=IFF+1
      96(IFF-K*(IFF/2).NE.1) GO TO 35
      DO 30 J=1,N
30    A(IRW,J)=0
      GO TO 3
35    CONTINUE
      DO 40 J=1,N
40    A(J,IRW)=0
      GO TO 3
25    CONTINUE
      DO 45 I=1,N
      DO 45 J=/=5
45    A(I,J)=DUM(I,J)
      PRINT 106
      DO 50 I=1,N
50    PRINT 104,(A(I,J),J=1,N)
101   FORMAT )I2)
102   FORMAT (//1X,* THE INPUT MATRIX IS...*,//)
103   FORMAT (15I1)
104   FORMAT (/X,15IB&
10    6OR4IT (/1,1K.*3BE-954D354 B 54-F9UTH DATRI7*=01&
106   FORMAT (//,1X,*THE MATRIX IN DESIRED BAND DIAGONAL FORM
     1 AFTER ROW AND COL. REMOVAL*,//)
107   FORMAT (/1X,* THE NEW SEQUENCE OF ROWS IS...*,/1X,15I2)
```

```
      STOP
      END
      SUBROUTINE ROWINC(A,B,N,DR)
C==
C===  THIS ROUTINE FINDS ROW INCIDENCES AND FILLS ENTRIES IN
C===  UPPER TRIANGULAR PORTION OF B.
C===
      INTEGER A(15,15),B(15,15),DR(15)
      N1=N-1
      46 5 I=T£51
      I1=I+1
      DO 10 K.I1,N
      DO 15 J=1,N
      IF(A(I,J).EQ.0) GO TO 15
      IF(A(I,J).NE.A(K,J)) GO TO 15
      DR(I)=DR(I)+1
      DR(K)=DR(K)+1
      B(I,K)=1
      GO TO 10
 15   CONTINUE
 10   CONTINUE
 5    CONTINUE
      RETURN
      END
      2UBROUTIE5 COLINC(A,B,N,DC)
C===
C===  THIS ROUTINE FINDS COL. INCIDENCES AND FILLS ENTRIES IN
C===  LOWER TRIANGULAR PORTION OF B.
C===
      INTEGER A(15,15),B(15,15),DC(15)
      N1=N-1
      DO 5 J=1,N1
      J1=J+1
      DO 10 L£J1,N
      DO 15 I=1,N
      IF(A(I,A).EQ.0) GO TO 15
      IF(A(I,J).EE.A(I,L)) GO TO 15
      DC(L)=DC(L)+1
      DC(J)=DC(J)+1
      B(L,J)=1
      GO TO 10
 15   CONTINUE
 10   CONTINUE
 5    CONTINUE
      RETURN
      END
      SUBROUTINE NMBRWS(A,B,N,R,DR,SR,TAGR,ICL,SSR)
C===
C===  THIS ROUTINE PERMUTES ROWS/COLS. USING TEWARSON&S ALGORITHM.
C===  NOTATIONS USED ARE AS FOLLOWS
C===  KR= NO.OF ROWS OF ZERO DEGREE
C===  9NDEX= VARIABLE SHOWING HOW MANY ROWS/COLS. ARE NUMBERED
C===  NGHBR= NEIGHBOUR VECTOR OF A NODE(NUMBERED), WHICH
C===  ARE CONSIDERED FOR LABELLING.
```

```
C---  INEX= VECTOR CONTAINING NODES WHICH ARE CONSIDERED FOR
C---  FINDING INTERSECTIONS WITH HIGHEST LABELLED NODE.
C--   3ORCOL= NEIGHBOUR COLS. OF A COL. ALREADY NUMBERED, WHICH
C---  ARE BEING TESTED FOR FIRST ROW-CODE PRESENCE.
C---  IN= SHOWS NO. OF NEIGHBOURS OF A NODE BEING CONSIDERED.
C --- IFL AND IFLR= FLAGS
C -   5D29W= NF= OF R6W FBI3H 1PPEARS FIRST IN ROW 36452
C --- ALREADY FOUND.
C---
      INTEGER A(15,15),B(15,15),R(15),DR(15),SR(15),TAGR(15),
     1 NGHBR(15),INEX(15),SSR(15),CORCOL(15),NMBRW(15)
      KR=0
      IFLG=0
      INDEX=0
      IN=0
      DO 1 I=1,N
 1    NGHBR(I)=0
 2    IFL=0
      CALL MNDGRW(TAGR,DR,R,N,IFL,IR)
C---
C---  IFL=1 IMPLIES ONLY ONE ROW OF MIN. DEGREE IS FOUND.
C---
      9F(IFL=EH=A& GO TO 5
      3ALL MNRWSM(TAGR,R,N,IR)
 5    CONTINUE
C---
C---  IR CONTAINS ROW NUMBER
C---
      9F(DR(9I)=NE.0) GO TO 10
      IZ=N-KR
      SR(IZ)=IR
      TAGR(IR)=1
      KR=KR+1
      GO TO 2
 6    IB=1
      INDEX=1
      GO TO 40
 10   CONTINUE
      IF(ICL.EQ.1) GO TO 6
      INDEX=INDEX+1
      SR(INDEX)=IR
      TAGR(IR)=1
      IF(INDEX.EQ.N-KR) RETURN
      IB=INDEX+1
 11   CONTINUE
      CALL MODIFY(N,B,IR,IN,NGHBR,DR,TAGR)
 12   CONTINUE
      IF(NGHBR(1).EQ.0) GO TO 40
      IF(IN.EQ.1) GO TO 16
      CALL MINDGR(NGHBR,IN,DR,IFLR,IR,MIN)
      IF(IFLR.EQ.1) GO TO 15
      IF(ICL.EH.1) GO TO 45
      CALL INTHLN(INDEX,SR,A,IN,NGHBR,MIN,N,IR,KOUNT,INEX,DR)
      IF(KOUNT.EQ.1) GO TO 15
```

```
      CALL MNEXOR(N,KOUNT,INEX,INDEX,SR,R,IR,A)
      GO TO 15
16    IR=NGHBR(1)
15    INDEX=INDEX+1
      SR(INDEX)=IR
      TAGR(IR)=1
      IF(INDEX.EQ.N-K9) RETURN
      IF(IN.EQ.1) GO TO 30
      DO 20 I=1,IN
      IF(NGHBR(I).NE.IR) GO TO 20
      IR1=I
      GO TO 25
20    CONTINUE
25    IR1=IR1+1
      IF(IR1.GT.IN) GO TO 30
      DO 35 I=IR1,IN
      IM=I-1
35    NGHBR(IM)=NGHBR(I)
30    NGHBR(IN)=0
      IN=IN-1
      IF(IN.EQ.0) GO TO 39
      CALL MDFYRW(IN,NGHBR,B,DR,IR)
      GO TO 12
39    DR(IR)=0
40    IR=SR(IB)
      IB=IB+1
      GO TO 11
45    CONTINUE
C---
C---  PROGRAM COMES HERE ONLY WHEN COLS. ARE BEING PERMUTED.
C---
      CALL FRSRWC(IN,NGHBR,SSR,MIN,N,DR,A,NMBRW,IR,KOUNT,IK,
     1CORCOL,IMN)
      IF(KOUNT.EQ.1) GO TO 50
      CALL MINRCD(IK,CORCOL,IMN,NMBRW,SSR,IR,N,A)
50    CONTINUE
      GO TO 15
      END
      SUBROUTINE MNDGRW(TAGR,DR,R,N,IFL,IR)
C---
C---  THIS ROUTINE FINDS NODE WHOSE DEGREE IS MINIMUM. IF
C---  CLASHES OCCUR, IFL=0.
C---
      INTEGER TAGR(15),DR(15),R(15)
      MIN=20
      DO 5 I=1,N
      IF(TAGR(I).EQ.1) GO TO 5
      IF(MIN.GT.DR(I)) MIN=DR(I)
5     CONTINUE
      KOUNT=0
      DO 10 I=1,N
      IF(TAGR(I).EQ.1) GO TO 10
      IF(MIN.NE.DR(I)) GO TO 10
      KOUNT=KOUNT+1
```

```
      IR=I
 10   CONTINUE
      IF(KOUNT.GT. ) RETURN
      IFL=1
      RETURN
      END
      SUBROUTINE MNRWSM(TAGR,R,N,IR)
C---
C---  THIS ROUTINE FINDS OUT ROW WHOSE ROW SUM IS MINIMUM.
C---
      INTEGER TAGR(15),R(15)
      MINR=20
      DO 5 I=1,N
      IF(TAGR(I).EQ.1) GO TO 5
      IF(MINR.GT.R(I)) MINR=R(I)
 5    CONTINUE
      DO 10 I=1,N
      IF(TAGR(I).EQ.1) GO TO 10
      IF(R(I).NE.MINR) GO TO 10
      IR=I
      RETURN
 10   CONTINUE
      SNBROUTINE MODIFY(N,B,IR,IN,NGHBR,DR,TAGR)
C---
C---  THIS ROUTINE MODIFIES INCIDENCE MATRIX OF ROW-NODE FOR
C---  THAT ROW WHICH IS ALREADY NUMBERED.
C---
      INTEGER B(15,15),NGHBR(15),DR(15)
      INTEGER TAGR(15)
      DR(IR)=0
      DO 5 I=1,N
      IF(TAGR(I).EQ.1) GO TO 5
      IF(I.GE.IR) GO TO 10
      IF(B(I,IR).EQ.0) GO TO 5
      IN=IN+1
      NGHBR(IN)=I
      B(I,IR)=0
      DR(I)=DR(I)-1
 5    CONTINUE
 10   CONTINUE
      IF(IR.EQ.N) RETURN
      IR1=IR+1
      DO 15 I=IR1,N
      IF(TAGR(I).EQ.1) GO TO 15
      IF(B(IR,I).EQ.0) GO TO 15
      IN=IN+1
      NGHBR(IN)=I
      B(IR,I)=0
      DR(I)=DR(I)-1
 15   CONTINUE
      RETURN
      END
      SUBROUTINE MINDGR(NGHBR,IN,DR,IFLR,IR,MIN)
C---
```

```
C===  THIS ROUTINE CHOSES MINIMUM DEGREE ROW FROM AMONG THE
C===  NEIGHBOURS OF ALREADY NUMBERED ROW.
C===
      INTEGER NGHBR(15),DR(15)
      IFLR=0
      MIN=20
      DO 5 I=1,IN
      NR=NGHBR(I)
      IF(MIN.GT.DR(NR)) MIN=DR(NR)
 5    CONTINUE
      KOUNT=0
      DO 10 I=1,IN
      NR=NGHBR(I)
      IF(MIN.NE.DR(NR)) GO TO 10
      KOUNT=KOUNT+1
      IR=NR
 10   CONTINUE
      IF(KOUNT.GT.1) RETURN
      IFLR=1
      RETURN
      END
      SUBROUTINE INTHLN(INDEX,SR,A,IN,NGHBR,MIN,N,IR,KCUNT,INEX,DR)
C==
C===  IN CASE WHEN TWO NEIGHBOURS OF A ROW-NODE HAVE SAME
C===  REMAINING DEGREE, THIS ROUTINE IS CALLED TO CALCULATE
C===  INTERSECTIONS OF THESE NEIGHBOURS WITH HIGHEST LABELLED
C===  NODE, AND SELECT IF POSSIBLE, ONE HAVING MAX. INTERSECTIONS
C===
      INTEGER A(15,15),SR(15),NGHBR(15),IINT(15),NBR(15),INEX(15)
      INTEGER DR(15)
      IHLN=SR(INDEX)
      INR=0
      DO 1 I=1,N
      NBR(I)=0
      INEX(I)=0
    1 IINT(I)=0
      DO 5 I=1,IN
      NR=NGHBR(I)
      IF(MIN.NE.DR(NR)) GO TO 5
      INR=INR+1
      NBR(INR)=NR
      DO 10 J=1,N
      IF(A(NR,J).EQ.0) GO TO 10
      IF(A(IHLN,J).NE.A(NR,J)) GO TO 10
      IINT(INR)=IINT(INR)+1
 10   CONTINUE
 5    CONTINUE
      MAX=0
      DO 15 I=1,INR
      IF(MAX .LT. IINT(I))  MAX=IINT(I)
 15   CONTINUE
C===
C===  MAX- MAXIMUM INTERSECTION BETWEEN ROWS.
C===
```

```
      KOUNT=0
      DO 20 I=1,INF
      IF(IINT(I) .NE. MAX)  GO TO 20
      KOUNT=KOUNT+1
      IR=NBR(I)
      INEX(KOUNT)=NBR(I)
 20   CONTINUE
      RETURN
      END
      SUBROUTINE MNEXOR(N,KOUNT,INEX,INDEX,SR,R,IR,A)
C===
C===  THIS ROUTINE FINDS EXCLUSIVE ORS BETWEEN ROWS HAVING
C===  CLASHES AND ROWS ALREADY NUMBERED IN REVERSE ORDER.
C===  WHENEVER A CLASH IS RESOLVED, RETURNS WITH NO. OF ROW.
C===
      INTEGER INEX(15),EXOR(15),SR(15),R(15)
      INTEGER A(15,15)
      DO 1 I=1,KOUNT
 1    EXOR(I)=0
      INDEX1=INDEX
 2    IQ=SR(INDEX1)
      DO 5 I=1,KOUNT
      IP=INEX(I)
      KOUNT1=0
      DO 10 J=1,N
      IF(A(IP,J).EQ.0) GO TO 10
      IF(A(IP,J).NE.A(IQ,J)) GO TO 10
      KOUNT1=KOUNT1+1
 10   CONTINUE
 5    EXOR(I)=R(IP)+R(IQ)-2*KOUNT1
      MN=20
      DO 15 I=1,KOUNT
      IF(MN.GT.EXOR(I)) MN=EXOR(I)
 15   CONTINUE
      KOUNT2=0
      DO 20 I=1,KOUNT
      IF(EXOR(I).NE.MN) GO TO 20
      IR=INEX(I)
      KOUNT2=KOUNT2+1
 20   36NTINUE
      IF(KOUNT2.EQ.1) RETURN
      INDEX1=INDEX1-1
      IF(INDEX1.NE.0) GO TO 2
      IR=INEX(1)
      RETURN
      END
      SUBROUTINE MDFYRW(IN,NGHBR,B,DR,IR)
C===
C===  THIS ROUTINE MODIFIES MATRIX B IN SO MUCH THAT ADJACENCY
C===  6F ONLY THOSE ROW-NODES, EXISTING IN NGHBR VECTOR,
C===  IS AFFECTED.
C===
      INTEGER NGHBR(15),B(15,15),DR(15)
      DR(IR)=0
```

```
      DO 5 I=1,IN
      NR=NGHBR(I)
      IF(NGHBR(I).GT.IR) GO TO 10
      IF(B(NR,IR).EQ.0) GO TO 5
      DR(NR)=DR(NR)-1
      B(NR,IR)=0
      GO TO 5
 10   CONTINUE
      IF(B(IR,NR).EQ.0) GO TO 5
      DR(NR)=DR(NR)-1
      B(IR,NR)=0
 5    CONTINUE
      RETURN
      END
      SUBROUTINE FSCOL(SR,A,N,IR,TAGC)
C---
C---  THIS ROUTINE FINDS THAT COL. OF A WHICH WILL APPEAR
C---  FIRST IN FINAL MATRIX.
C---
      INTEGER SR(15),A(15,15),COLS(15),TAGC(15)
      IA=0
      DO 1 I=1,N
 1    COLS(I)=0
      ISR=SR(1)
      DO 5 J=1,N
      IF(A(ISR,J).EQ.0) GO TO 5
      IA=IA+1
      COLS(IA)=J
      TAGC(J)=1
 5    CONTINUE
      IB=IA
      DO 10 I=1,N
      IF(I.EQ.ISR) GO TO 10
      DO 15 J=1,IA
      IC=COLS(J)
      IF(A(I,IC).EQ.0) GO TO 15
      DO 20 K=1,N
      IF(A(I,K).EQ.0) GO TO 20
      IF(TAGC(K).NE.0) GO TO 20
      IB=IB+1
      COLS(IB)=K
      TAGC(K)=1
 20   CONTINUE
 15   CONTINUE
 10   CONTINUE
      IA=IB
      IF(IA.EQ.1) GO TO 100
      DO 25 I=1,N
      II=N-I+1
      ISR=SR(II)
      DO 30 J=1,N
      IF(A(ISR,J).EQ.0) GO TO 30
      IF(TAGC(J).EQ.0) GO TO 30
      IA=IA-1
```

```
      TAGC(J)=0
   30 CONTINUE
      IF(IA.EQ.1) GO TO 100
      IF(IA.EQ.0) GO TO 35
   25 CONTINUE
C---  PROGRAM NEVER COMES HERE  FROM 25 DOWNWARDS.
   35 CONTINUE
      IQ=0
      ISR=SR(1)
      DO 40 J=1,N
      IF(A(ISR,J).EQ.0) GO TO 40
      IQ=IQ+1
      COLS(IQ)=J
      TAGC(J)=1
   40 CONTINUE
      IF(IQ.EQ.1) GO TO 100
      IA=IQ
      DO 45 I=2,N
      IST=SR(I)
      DO 50 J=1,IQ
      IC=COLS(J)
      IF(A(IST,IC).NE.0) GO TO 50
      TAGC(IC)=0
      IA=IA-1
      IF(IA.EQ.1) GO TO 100
   50 CONTINUE
   45 CONTINUE
C---  PROGRAM NEVER COMES HERE FROM 45 DOWMWARDS.
  100 CONTINUE
      DO 55 I=1,N
      IF(TAGC(I).EQ.0) GO TO 55
      IR=I
      RETURN
   55 CONTINUE
      SUBROUTINE NMBCOL(A,B,N,F,DC,SC,TAGC,ICL,C,SR)
C---
C---  THIS ROUTINE TRANSPOSES A AND THEN CALLS NMBRWS TO
C---  RENUMBER ROWS(WHICH ARE COLS. OF ORIGINAL MATRIX)
C---
      INTEGER A(15,15),B(15,15),F(15,15),SR(15),SC(15),
     1 C(15),DC(15),TAGC(15)
      ICL=1
      DO 5 I=1,N
      DO 5 J=1,N
    5 A(I,J)=F(J,I)
      CALL NMBRWS(A,B,N,C,DC,SC,TAGC,ICL,SR)
      PRINT 101,(SC(I),I=1,N)
  101 FORMAT (/1X,* THE NEW COLUMN SEQUENCE IS...*,/1X,15I2)
      RETURN
      END
      SUBROUTINE FRSRWC(IN,NGHBR,SSR,MIN,N,DR,A,NMBRW,IR,
     1 KOUNT,IK,CORCOL,IMN)
C---
C---  THIS ROUTINE FINDS THAT COL. WHOSE ROW-CODE APPEARS
```

```
C---   FIRST IN SORTED ROW VECTOR.
C---
       INTEGER NGHBR(15),SSR(15),DP(15),A(15,15),
      1 NMBRW(15),CORCOL(15)
       IK=0
       DO 1 I=1,N
       CORCOL(I)=0
 1     NMBRW(I)=0
       DO 5 I=1,IN
       NR=NGHBR(I)
       IF(MIN.NE.DP(NR)) GO TO 5
       DO 10 J=1,N
       ISR=SSR(J)
       IF(A(NR,ISR).NE.1) GO TO 10
       IK=IK+1
       NMBRW(IK)=J
       CORCOL(IK)=NR
 10    CONTINUE
 5     CONTINUE
       IMN=20
       DO 15 I=1,IK
       IF(IMN.GT.NMBRW(I)) IMN=NMBRW(I)
 15    CONTINUE
       KOUNT=0
       DO 20 I=1,IK
       IF(NMBRW(I).NE.IMN) GO TO 20
       IR=CORCOL(I)
       KOUNT=KOUNT+1
 20    CONTINUE
       RETURN
       END
       SUBROUTINE MINRCD(IK,CORCOL,IMN,NMBRW,SSR,IR,N,A)
C---
C---   THIS ROUTINE CALCULATES MAX. ROW-CODE DISTANCE BETWEEN
C---   COLS. WHICH HAVE CLASHED ON FIRST ROW-CODE APPEARANCE COUNT.
C---
       INTEGER CORCOL(15),NMBRW(15),SSR(15),A(15,15),
      1 DIFF(15),RENUM(15)
       IT=0
       DO 1 I=1,N
       DIFF(I)=0
 1     RENUM(I)=0
       DO 5 I=1,IK
       IF(NMBRW(I).NE.IMN) GO TO 5
       NN=CORCOL(I)
       DO 10 J=1,N
       II=N-J+1
       ISR=SSR(II)
       IF(A(NN,ISR).NE.1) GO TO 10
       IT=IT+1
       DIFF(IT)=II-NMBRW(I)
       RENUM(IT)=NN
 10    CONTINUE
 5     CONTINUE
```

```
      MIN=20
      DO 15 I=1,IT
      IF(MIN.GT.DIFF(I)) MIN=DIFF(I)
 15   CONTINUE
      DO 20 I=1,IT
      IF(DIFF(I).NE.MIN) GO TO 20
      IR=RENUM(I)
      RETURN
 20   CONTINUE
      END
      SUBROUTINE RWCLRM(A,MAXM,N,IRW,IFF)
C---
C---  THIS ROUTINE CALCULATES BAND-WIDTH OF A AND CALLS ROUTINE
C---  WHICH REMOVES ROWS AND COLS. OF A TO REDUCE ITS BW.
C---
      INTEGER A(15,15),D(15,15),BW
      IRW=0
      NCLFR=0
      NRWFC=0
C---
C---  NCLFR= NO. OF COLS. IN FIRST ROW
C---  NRWFC= NO. OF ROWS IN FIRST COL.
C---
      DO 5 I=1,N
      DO 10 J=1,I
      IF(A(I,J).EQ.0) GO TO 10
      IFR=I-J+1
      IF(IFR.GT.NRWFC) NRWFC=IFR
      GO TO 5
 10   CONTINUE
 5    CONTINUE
      N1=N-1
      DO 15 I=1,N1
      DO 20 J=I,N
      JJ=N-(J-I)
      IF(A(I,JJ).EQ.0) GO TO 20
      ILS=JJ-I+1
      IF(ILS.GT.NCLFR) NCLFR=ILS
      GO TO 15
 20   CONTINUE
 15   CONTINUE
      BW=NRWFC+NCLFR-1
      PRINT 101,BW
      IF(BW.EQ.MAXM) RETURN
      IR1=NRWFC
      IC1=NCLFR
      IF(IFF-2*(IFF/2).NE.0) GO TO 25
      CALL RWRMVL(A,N,IR1,IC1,IRW,IFF)
      RETURN
 25   CONTINUE
      DO 30 I=1,N
      DO 30 J=1,N
 30   D(I,J)=A(I,J)
      DO 35 I=1,N
```

```
      DO 35 J=1,N
 35   A(I,J)=D(J,I)
      CALL RWRMVL(A,N,IC1,IR1,IRW,IFF)
      DO 40 I=1,N
      DO 40 J=1,N
 40   A(I,J)=D(I,J)
 101  FORMAT (/1X,* THE BAND-WIDTH =*,I2)
      RETURN
      END
      SUBROUTINE RWRMVL(A,N,IR1,IC1,IRW,IFF)
C----
C----  THIS ROUTINE REMOVES ROWS/COLS. OF A TO REDUCE
C----  ITS BAND-WIDTH.
C----
      INTEGER A(15,15)
      IQ=IR1+IC1
      DO 5 I=1,IR1
      IJ=I+IC1-1
      IF(IJ.GT.N) IJ=N
      IF(A(I,IJ).NE.1) GO TO 5
      IF(IJ.EQ.N) GO TO 20
      JIN=N-IJ
      I1=I
      DO 25 K=1,JIN
      I1=I1+1
      J1=IJ+K
      IF(A(I1,J1).EQ.1) GO TO 5
 25   CONTINUE
 20   IRW=I
      GO TO 30
 5    CONTINUE
      IF(IR1.EQ.N) GO TO 30
      IF(IR1+IC1.GE.N) GO TO 35
      IF(A(IR1,1).NE.0) GO TO 30
      IR2=IR1
      IC2=IR1+IC1-1
      N1=N-1
      IK=1
      DO 45 I=IQ,N1
      IR2=IR2+1
      IC2=IC2+1
      IK=IK+1
      IF(A(IR2,IK).NE.1.OR.A(IR2,IC2).NE.1) GO TO 50
      IR3=IR2
      IC3=IC2+1
      DO 55 J=IC3,N
      IR3=IR3+1
      IF(A(IR3,J).NE.0) GO TO 45
 55   CONTINUE
      IRW=IR2
      GO TO 30
 50   CONTINUE
      IF(A(IR2,IK).NE.0) GO TO 35
 45   CONTINUE
```

```
 35    CONTINUE
       IR2=N-IC
       IF(IQ.GE.N) IR2=IR1
       IC2=N+1-IR1-IC1
       IF(IQ.GE.N) IC2=1
C---
C---   IR2- NO. OF ROWS ALREADY CONSIDERED.
C---   IC2- NO. OF COLS. TO BE LEFT NOW.
C---
       IF(IR2.EQ.N) GO TO 30
       ID=0
       DO 60 I=IR1,IR2
       ID=ID+1
       IF(A(I,ID).NE.0) GO TO 30
 60    CONTINUE
       IG=N-IR2
       DO 70 I=1,IG
       IR2=IR2+1
       IC2=IC2+1
       IF(A(IR2,IC2).NE.1.OR.A(IR2,N).NE.1) GO TO 75
       IRW=IR2
       GO TO 30
 75    CONTINUE
       IF(A(IR2,IC2).NE.0) GO TO 30
 70    CONTINUE
 30    CONTINUE
       IF((IFF-(IFF/2)*2).EQ.1) GO TO 80
       PRINT 101,IRW
       RETURN
 80    PRINT 102,IRW
 101   FORMAT (//1X,* THE REMOVED ROW IS*,I2)
 102   FORMAT (//1X,* THE REMOVED COLUMN IS*,I2)
       RETURN
       END
£ENTRY
07
0100000
0010000
0000110
0111000
0001010
1000010
0000101
```

THE COMPUTER OUTPUT FOR THE TEST-DATA IS...

THE INPUT MATRIX IS...

```
0 1 0 0 0 0 0
0 0 1 0 0 0 0
0 0 0 0 1 1 0
0 1 1 1 0 0 0
0 0 0 1 0 1 0
```

REFERENCES

1. Alway G.G. and Martin D.W. - 'An algorithm for reducing the Bandwidth of a matrix of symmetrical configuration', Computer J., vol.8 (1965-66) 264-272.

2. Brayton R.K., Gustavson F.G. and Willoughby R.A. - 'Some results on sparse matrices', IBM Thomas J. Watson Research Centre, Feb. 1969, RC 2332.

3. Edwards C.R. - 'The Logic of Boolean matrices', Computer J., Vol. 15 (1972) 247.

4. Fine N.J. - 'On the Walsh functions', Trans. Am.Math. Soc., Vol.65 (1949) 372-414.

5. Garwick J. - 'ALGOL Programming (Section) : Contribution 7. Solution of a linear system with a Band coefficient matrix', BIT, Vol.3 (1963) 207-208.

6. Golomb S. - 'SHIFT REGISTER SEQUENCES,' San Fransisco, Holden.Day, 1967.

7. Hammer P.L. and Rudeaunu S. - 'BOOLEAN METHODS IN OPERATIONS RESEARCH', Springer Verlac, N.Y., 1968.

8. Harary F. - 'Graphs and matrices', SIAM Rev., Vol.9, No.1, (Jan.'67) 83-90.

9. - 'Sparse matrices and Graph Theory', in 'Large Sparse Sets of Linear Equations', (ed. J.K. Reid) 139-150.

10. Harmuth H.F. - 'TRANSMISSION OF INFORMATION BY ORTHOGONAL FUNCTIONS' Springer Verlac, N.Y., 1969.

11. Jennings A. - 'A Compact Storage Scheme for the Solution of Symmetric linear simultaneous equations', Computer J., Vol.9 (1966-67) 281-285.

12. Ledley R.S. - 'DIGITAL COMPUTER AND CONTROL ENGINEERING', McGraw Hill, N.Y., 1960.

13. Livesley R. - 'An analysis of large structural system', Computer J., Vol.3, No.1 (April, 1960) 34-39.

14. Lo Dato V.A. - 'The permulation of certain class of matrices', Computer J., Vol. 13, No.4 (Nov., 1970) 405-; -

15. Martin R.S. and Wilkinson J.H. - 'Symmetric decomposition of positive definite band matrices', Numer.Math., Vol.7 (1965) 355-361.

16. McCormick W.A. - 'Application of partially banded matrix methods to structural analysis', in Reference 41 pp 25-34.

17. Oberman R.M. - 'DISCIPLINES IN COMBINATORIAL AND SEQUENTIAL CIRCUIT DESIGN' McGraw Hill, N.Y., 1970.

18. Ogbuobiri E.C. - 'Dynamic storage and retrieval in sparsity programming', IEEE Trans. PAS - 89 (1970b) 150-155.

19. Palacol E.L. - 'The finite element method of structural analysis', in Reference (41), pp 101-105.

20. Paley R.E.A.C. - 'A remarkable series of orthogonal functions', Proc. London Math.Soc., Vol.2, No.34, 241-279.

21. Pooch and Nieder - 'A survey of indexing techniques for sparse matrices', ACM Computing Surveys, Vol.5, No.2 (1973) 109 - - -.

22. Rajaraman V. and Muthukrishnan C.R. - 'Algorithms to partition Boolean matrices', Computer Centre, I.I.T. Kanpur.

23. Ralston A. - 'Numerical integration methods for the solution of ordinary differential equations' in Mathematical Methods for Digital Computers, (Eds.-A. Ralston and A.S. Wilf), Vol.I, John Wiley and Sons, N.Y. 1967, pp 95-109.

24. Romanelli M. - 'Runge-Kutta methods for the solution of ordinary differential equations', Ibid; pp.110-120.

25. Rosen J.B. - 'Primal, partitioning programming for block diagonal matrices, Numer.Math., Vol.6(1964), 250-260.

26. Rosenberg - 'METHODS FOR THE NUMERICAL SOLUTION OF PARTIAL DIFFERENTIAL EQUATIONS' American Elsevier, N.Y., 1969.

27. Steward D.V. - 'On an approach to techniques for the analysis of the structure of large systems of equations', SIAM Rev., Vol.4, No.4 (Oct.1962) 321-342.

28. - 'Tearing analysis of the structure of disorderly sparse matrices', in Reference (41), pp 65-74.

29. - 'Partitioning and tearing systems of equations', SIAM J. Numer. Anal. Series B., Vol.2, No.2 (1965) 345-365.

30. Tewarson R.P. - 'Row-column permutation of sparse matrices', Computer J., Vol.10 (1967) 300-305.

31. - 'SPARSE MATRICES', Academic Press, N.Y. and London, 1973.

32. - 'On the product form of inverse of sparse matrices', SIAM Rev., Vol.8 (1966) 336-342.

33. - 'Computations with sparse matrices', SIAM Rev. 12, 4(Oct. 1970) 527-543.

34. - 'Solution of linear equations with coefficient matrix in band form', BIT, Vol.8 (1968) 53-58.

35. Thurnan D. - 'Algorithm 195 ; BANDSOLVE', CACM, Vol.6, No.8 (Aug. 1963) 441---.

36. Tinney W.F. - 'Comments on using sparsity techniques for power system problems', in Reference (41), pp 25-34.

37. Wachspress E. - 'The numerical solution of boundary value problems', in Mathematical Methods for Digital Computers (Eds. A. Ralston and A.S. Wilf), Vol.I, John Wiley, N.Y. 1967, pp 121-127.

38. Walsh J.L. - 'A closed set of orthogonal functions', Am.J. Math., Vol.45 (1923) 5-24.

39. Weil R.L. and Kettler P.C. - '

Management Science, 18, 1 (Sept. 1971) 98-108.

40. - 'An algorithm to provide structure for decomposition', in Reference (41), pp 11-24.

41. Willoughby R.A. (Ed.) - 'Sparse matrix proceedings', Held at IBM T.J. Watson Research Centre, March 1969, RA 11707.

42. - 'A survey of sparse matrix technology', IBM T.J.W. Research Centre, May 1972, RC 3872.

43. - 'Sparse matrix algorithms and their relation to problem classes and computer architecture', IBM T.J.W. Research Centre, March 1970, RC 2833.

44. Yuen C.K. - 'Walsh functions for the computer scientist', Dept. of Computer Science, Univ. of Sydney.

45. - 'Walsh functions and Gray code', IEEE Trans. EMC-15 No.3 (1971) 68-73.

46. IEEE Trans. on EMC, Vol. 11-13, Symposium on Application of Walsh Transforms.

47. 1972 and 1973 Proceedings of Symposium on Application of Walsh Transforms.